2011/10/29

الجمهورية العربية السورية – دمشق- معضمية الشام

الصفحة على الفيسبوك :

WWW.FACEBOOK.COM/KHALEDYASSINKH

الإجراءات و التوابع و العودية(التراجعية)

بنى التحكم و المعطيات البسيطة والمعطيات المركبة و السجلات و المصفوفات(الأنساق) و نوع المعطيات كائن object والمجموعات
حل مسائل.

الجزء الأول

Syrian Arab Republic

ومن يتهيب صعود الجبال
يعش أبد الدهر بين الحفر

إعداد المهندس خالد ياسين الشيخ

بسم الله الرحمن الرحيم

الإجراءات و التوابع procedure and functions

ملاحظة:لغة باسكال لا تميز بين الحروف الصغيرة و الحروف الكبيرة في تسمية المتحولات و الإجراءات والتوابع

الصفات العامة للإجراءات procedure general features:

يتكون الإجراء من مجموعة من العبارات المتجمعة فيما بينها لإنجاز عمل جزئي معين و اسم الإجراء يُشير إلى هذه المجموعة من العبارات و في أغلب الأحيان يكون الإجراء عبارة عن برنامج مُصغر يمكن أن يملك قسم تصريح خاص به أو إجراءاته الخاصة به أيضاً.

لقد صممت الإجراءات لإنجاز بعض المهام الجزئية مثل مسح الشاشة أو ترتيب مجموعة من القيم أو إخراج صوت عندما يريد المستخدم ذلك وهكذا.... ، و لكن قد نحتاج في بعض الأحيان إلى إجراء ينجز بعض الأعمال العامة مع بعض التغييرات في كل مرة ينفذ فيها هذا الإجراء فعلى سبيل المثال: إذا أخذنا إجراء يجمع مجموعة من الأعداد فإنه في كل مرة يُستدعى فيها هذا الإجراء ستكون هذه الأعداد و القيمة الناتجة عن جمع هذه الأعداد مختلفة عن المرة السابقة و هكذا... لذلك يجب على الإجراءات أن تكون قادرة على التعامل مع هذه التغييرات حتى تمكن من إنجاز أعمالها كما ينبغي. و حتى تتمكن الإجراءات من التعامل مع هذه التغييرات يجب أن تكون قادرة على الاتصال مع الروتين المستدعي (البرنامج الرئيسي أو روتين آخر).

تملك الإجراءات بشكل عام طريقتين للاتصال مع الروتين المستدعي هما:

1. بواسطة المتحولات global variables إذ تستخدم هذه المتحولات ضمن أي جزء من أجزاء البرنامج.
2. بواسطة الوسطاء parameters و التي يكن عن طريقها إمرار المعلومات من و إلى الإجراء.

تعريف الإجراء و استدعائه procedure definitions and Calls:

من المهم أولاً التمييز بين تعريف الإجراء و استدعائه فمن أجل استخدام الإجراء يجب تعريفه أولاً ثم استدعائه من خلال البرنامج. و يتم تعريف الإجراء بتحديد المتحولات المستخدمة في هذا الإجراء ووضع سلسلة التعليمات التي بتنفيذها ينجز الإجراء العمل الموكل إليه. وتعريف الإجراء يجب أن يصرح عنه في قسم التصريحات الخاص بالبرنامج كما سنرى لاحقاً.
بعد أن نعرف الإجراء يصبح بإمكاننا استخدامه في برامجنا وعندما يُستدعى هذا الإجراء ينجز العمل المطلوب منه.ومما يجب ذكره أنه يجب علينا تعريف الإجراء قبل أن نتمكن من استدعائه ضمن البرنامج و استدعاء الإجراء هو عملية تقنية يتم من خلالها نقل تحكم البرنامج إلى الإجراء المستدعى فعندما نستدعي الإجراء ننفذ مجموعة التعليمات المحتواة في هذا الإجراء و بعد الانتهاء من تنفيذ آخر تعليمة من تعليمات الإجراء يُنقل التحكم إلى النقطة التي تأتي مباشرة بعد أمر استدعاء الإجراء في الروتين الذي استدعى الإجراء و البرنامج التالي يرينا كيف تتم هذه العملية:

code

```
program test;
var val1,val2:integer;

            procedure changeinfo;
             begin
(*7*)            val1:=val1*val2;
(*8*)            val2:=val1+val2;
(*9*)            val1:=val1*val2;
             end;
begin
(*1*) val1:=1;
(*2*) val2:=2;
(*3*) changeinfo;
(*4*) writeln(val1);
(*5*) writeln(val2);
(*6*) readln
end.
```

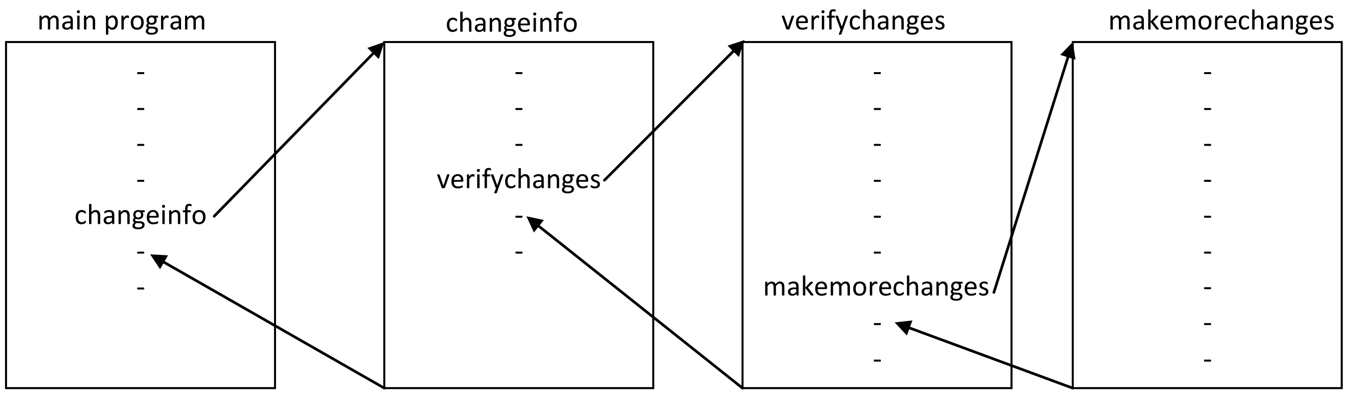
خرج البرنامج هو:

```
8
4
```

إن تسلسل تنفيذ التعليمات في البرنامج السابق يكون كما يلي:

1. يبدأ تنفيذ البرنامج من أول تعليمة في جسم البرنامج و هي السطر الأول في هذا البرنامج حيث تنفذ في هذا السطر و السطر الذي يليه عملية إلحاق قيم للمتحولات.
2. عندما يصل التنفيذ إلى السطر الثالث في البرنامج الرئيسي يتوقف تنفيذ تعليمات البرنامج الرئيسي بشكل مؤقت.
3. يُنقل التحكم إلى الإجراء changeinfo و بذلك تكون العبارة الواجب تنفيذها هي السطر السابع.
4. تُنفذ العبارات الموجودة في الإجراء changeinfo بنفس الموجودة فيه أي السطر السابع ثم الثامن ثم التاسع.
5. بعد الانتهاء من تنفيذ السطر التاسع يُعاد التحكم إلىثانية إلى البرنامج الرئيسي.
6. يستأنف البرنامج الرئيسي تنفيذ عباراته ابتداء من السطر الرابع.
7. ينتهي البرنامج بعد الانتهاء من تنفيذ السطرين الخامس و السادس.

هذه هي طريقة نقل التحكم بين البرنامج و الروتين (الإجراء) المستدعى من قبل هذا البرنامج.
إن استدعاءنا للإجراءات المسبقة التعريف مثل الإجراء writeln يتم بنفس الطريقة التي استدعينا بها الإجراء changeinfo يعني أن البرنامج الرئيسي يتوقف تنفيذه و بشكل مؤقت حتى ينفذ الإجراء writeln بشكل كامل ثم يُعاد التحكم إلى البرنامج الرئيسي و يتابع تنفيذ عباراته.
ليس شرطاً أن تُستدعى الإجراءات دائماً من قبل البرنامج الرئيسي وإنما يمكن للإجراءات أيضاً تستدعي بعضها ويُنقل التحكم بين هذه الإجراءات بشكل مشابه لما ذكرنا مع شيء من التعقيد.
فعلى سبيل المثال:يمكن أن تكون إحدى العبارات الموجودة في الإجراء changeinfo عبارة عن استدعاء آخر اسمه verifychanges و عند ذلك الاستدعاء يتوقف تنفيذ الإجراء changeinfo بشكل مؤقت و ينقل التحكم إلى الإجراء verifychanges أي أنه عند هذه النقطة يتوقف تنفيذ البرنامج الرئيسي و الإجراء changeinfo و بشكل مؤقت حتى ينتهي تنفيذ جميع عبارات الإجراء verifychanges و عندما ينتهي هذا الإجراء من عمله يعود التحكم ثانية إلى الإجراء changeinfo و الذي يستأنف تنفيذ عباراته ابتداء من التعليمة التالية لتعليمة استدعاء الإجراء verifychanges و بالطبع يمكن للإجراء verifychanges بدوره أن يستدعي إجراء آخر يسمى مثلاً makemorechanges و هكذا دواليك......



عندما ما يستدعى الإجراء يتم تحفيز مؤقت لبيئة موضعية environment خاصة به تتضمن تخزيناً مؤقتاً لكل المتحولات التي صرح عنها في الإجراء و هذه البيئة و التخزين المؤقت للمتحولات سوف تتلاشى بعد الانتهاء من تنفيذ مما يعني أن مواضع الذاكرة المخصصة لمتحولات الإجراء سوف تعاد إلى التجمع العام لحجر الذاكرة المخصصة للمتحولات العامة المستخدمة في البرنامج و ينتج عن هذا أنك لن تستطيع الوصول إلى مواضع تخزين المتحولات الخاصة بالإجراء عندما تتم مغادرة هذا الإجراء بعد الانتهاء من تنفيذه.

الإجراءات بلغة باسكال PASCAL Procedures:

إن البنية العامة للإجراء في لغة باسكال تتألف من الأجزاء التالية:

- رأس الإجراء A procedure Head
- قسم التصريح A Declaration Section
- جسم الإجراء A procedure Body

رأس الإجراء the procedure Head:

يتألف رأس الإجراء من المميز المحجوز procedure متبوعاً باسم الإجراء و الذي يمكن أن يكون أي مميز مسموح به في لغة باسكال و يمكن أن يُتبع اسم الإجراء بلائحة من الوسطاء و يجب أن ينتهي رأس الإجراء بفاصلة منقوطة لتفصله عن باقي أجزاء الإجراء.
وفيما يلي بعض الأمثلة المختلفة و المقبولة لرؤوس إجراءات:

1. PROCEDURE BeepUser;
2. Procedure wait;
3. procedure count(howhigh:integer);
4. procedure bubblesort(var info:string80array; size:integer);
5. procedure Getstring(message:string; value:string);
6. procedure GetDate(var Month,Date,Year:integer);

نلاحظ من خلال الأمثلة السابقة أن رأسي الإجراءين الأول و الثاني لا يملكان أي وسطاء أما رأس الإجراء الثالث فهو يملك وسيطاً واحداً ورأس الإجراءين الرابع و الخامس يملكان وسيطين لكل و هذه الوسطاء من أنواع مختلفة أما رأس الإجراء السادس فهو يملك ثلاثة وسطاء من نوع واحد.

قسم التصريح the Declaration Section:

يملك كل إجراء قسم خاص به إذ يمكن أن نصرح فيه عن ثوابت أو متحولات أو أنواع مختلفة أو نعرف فيه إجراءات و توابع أخرى.
وطريقة التصريح عن هذه الأنواع تخضع لنفس الطريقة المستخدمة في البرنامج الرئيسي مع الفوارق البسيطة في إمكانية الوصول إلى التعريفات و التصريحات.

جسم الإجراء the procedure body:

يحتوي جسم الإجراء على مجموعة من العبارات الضرورية ليقوم الإجراء بالعمل الموكل إليه. يبدأ جسم الإجراء بالمميز المحجوز begin و ينتهي بالمميز المحجوز end متبوعاً بفاصلة منقوطة مشيرة إلى أن الإجراء قد انتهت عباراته.وتوضع عبارات الإجراء كلها بين هذين المميزين المحجوزين.
عندما يُنقل التحكم إلى الإجراء يبدأ بتنفيذ عبارته الأولى و التي تأتي مباشرة بعد المميز المحجوز begin و يستمر هذا التنفيذ إلى أن يصل إلى آخر عبارة في جسم الإجراء و هي التي تأتي قبل المميز المحجوز end و بعدها يُعاد التحكم ثانية إلى الروتين المستدعي.

المتحولات العامة و المتحولات الموضعية Global and Local Variables:

المتحولات العامة Global Variables هي المتحولات التي يمكن استخدامها في البرنامج الرئيسي و في جميع الروتينات routines المعرفة ضمن البرنامج الرئيسي.
نستطيع أيضاً التصريح عن متحولات خاصة بكل إجراء في قسم التصريح لكل من الإجراءات و التوابع و هذه المتحولات تدعى عندها بالمتحولات الموضعية (المحلية):
البرنامج التالي يحتوي على متحولات عامة ضمن قسم التصريح للبرنامج الرئيسي و على متحولات موضعية ضمن أقسام التصريح في كل إجراء من إجراءاته المختلفة.
ومما تجب ملاحظته أن الثوابت و أنواع المعطيات المختلفة و حتى الروتينات تُعامل مثل المتحولات من حيث عموميتها و إمكانية استخدامها في الروتينات المعرفة داخل البرنامج.

code

```
program test;
const maxtrials=10000;
var Qtitle1,Qtitle2,Qtitle3,Qtitle4:real;
procedure randomsamples;
const
cutoff1=0.25;
cutoff2=0.50;
cutoff3=0.75;
var
result:real;
count:integer;
```

```
begin
writeln(' starting randomsamples');
randomize;
for count:=1 to maxtrials do
begin
result:=random;
if(result<cutoff1)then
Qtitle1:=Qtitle1+1
else
if(result<cutoff2)then
Qtitle2:=Qtitle2+1
else
if(result<cutoff3)then
Qtitle3:=Qtitle3+1
else
Qtitle4:=Qtitle4+1;
end;
writeln(' ending randomsamples');
end;
procedure displayresults;
const fwidth=8;
precision=4;
begin
writeln(' starting displayresults');
writeln(' Qtitle1 = ',qtitle1:fwidth:precision);
writeln(' Qtitle2 = ',qtitle2:fwidth:precision);
writeln(' Qtitle3 = ',qtitle3:fwidth:precision);
writeln(' Qtitle4 = ',qtitle4:fwidth:precision);
writeln(' ending displayresults');
end;
procedure BeepUser;
const BeepChar=^G;  (* ASSCI 7 *)

begin
writeln(' starting BeepUser');
write(BeepChar);
writeln(' Ending BeepUser');
end;
begin
writeln(' starting main program');
Qtitle1:=0;
Qtitle2:=0;
Qtitle3:=0;
Qtitle4:=0;
RandomSamples;
Displayresults;
BeepUser;
writeln(' ending main program');
readln
end.
```

خرج البرنامج شبيه بالخرج التالي:

```
starting main program
 starting randomsamples
 ending randomsamples
 starting displayresults
 Qtitle1 = 2480.0000
 Qtitle2 = 2481.0000
 Qtitle3 = 2530.0000
 Qtitle4 = 2509.0000
 ending displayresults
 starting BeepUser
 Ending BeepUser
 ending main program
```

يُولد البرنامج 10,000 قيمة عشوائية في المجال ما بين (0-1) إذ قُسم هذا المجال إلى أربعة مجالات جزئية متساوية ومهمة البرنامج حساب عدد الأعداد العشوائية المتولدة و الواقعة ضمن كل مجال جزئي.
وقد قُسم عمل هذا البرنامج إلى مجموعة من الإجراءات فالإجراء الأول randomsamples يولد عدداً عشوائياً و يفحص هذا العدد من حيث المجال الجزئي الذي ينتمي إليه ويزيد عداد الفئة التي يقع فيها هذا العدد ويكرر هذه العملية 10000 مرة
أما الإجراء الثاني Displayresults فيطبع اسم الفئات الجزئية مع عدد الأعداد المنتمية لكل فئة منها أما الإجراء الثالث BeepUser فيصدر صوت الجرس عند استدعائه.
يحتوي البرنامج السابق على متحولات عامة و متحولات موضعية أما المتحولات العامة فهي الثوابت والمتحولات المعرفة ضمن قسم التصريح في البرنامج الرئيسي (أربعة متحولات من النوع عدد حقيقي و ثابت واحد) أما المتحولات الموضعية فهي التصريحات الموجودة داخل كل إجراء.
السؤال الذي يطرح نفسه: ما هو الفرق بين المتحولات العامة و المتحولات الموضعية؟
المتحولات العامة :يمكننا استخدامها في أي مكان ضمن البرنامج الرئيسي لذا استطعنا استخدام المتحولات العامة Qtitle1,Qtitle2,Qtitle3,Qtitle4 ضمن الإجراءين randomsamples و displayResults أما المتحولات الموضعية فلا يمكن أن تُستخدم إلا ضمن الإجراء الذي عُرفت فيه حيث ينشى كل إجراء بيئة environment مُصغرة لتصريحاته و تعريفاته و إجراءاته الخاصة و تكون المتحولات الموضعية جزءاً من هذه البيئة فلا يمكن أن تتواجد هذه المتحولات خارج هذه البيئة.
مثلا:الثابتان Fwidth و precision مُعرفان ضمن الإجراء displayResults فإذا حاولت استخدامهما في البرنامج الرئيسي main program أو ضمن أحد الإجراءات فإن المترجم compiler سيعطيك رسالة الخطأ التالية Undeclared identifier و التي تشير إلى عدم إمكانية استخدام هذه الثوابت ضمن هذا الجزء من البرنامج و السبب في هذا أن هذه الثوابت غير معرفة و غير موجودة عندما لا يكون الإجراء displayresults فعالاً و بشكل مشابه فإن المتحول result لا يمكن استخدامه إلا ضمن الإجراء randomsamples فقط وهكذا............. .

الوسطاء :parameters

تزود المتحولات الوسيطية الإجراءات بالمعلومات اللازمة و ترجع النتائج المطلوبة من الإجراءات.فمثلاً :إجراء الترتيب يحتاج إلى معرفة عدد القيم الواجب ترتيبها و القيم المراد ترتيبها و نستطيع تمرير هذه المعلومات عن طريق المتحولات الوسيطية.
ورأس الإجراء يملك لائحة اختيارية من الوسطاء و هذه اللائحة هي البيئة environment التي يتم من خلالها إعداد المعلومات التي إعداد المعلومات التي سوف تمرر إلى الإجراء على أنها متحولات وسيطية.
يمكننا تجاوز اعتبار الوسطاء على أنها منافذ تُمرر من خلالها المعلومات إلى الإجراء و هذه المنافذ يجب أن تكون بأنواع محددة و يعتمد ذلك على نوع المعلومات المراد تمريرها إلى الإجراء و يمكن أن تمرر المعلومات باتجاه واحد أو باتجاهين أي إلى الإجراء أو من و إلى الإجراء .
عندما نُعرف الإجراء يكون بإمكاننا إضافة منفذ لكل متحول يجب أن يُمرر إلى الإجراء عندما يُستدعى الإجراء هذا الإجراء.
فعلى سبيل المثال يقوم الإجراء countvals في البرنامج التالي بالعد حتى عددٍ معين يمرر إليه عن طريق المنفذ الوحيد الموجود في لائحة الوسطاء لهذا الإجراء.

code

```
program procexample;
var nrtimes,i:integer;
procedure countvals(howhigh:integer);
var index:integer;
begin
for index:=1 to howhigh do
writeln(index);
end;
begin
countvals(28);
nrtimes:=6;
countvals(nrtimes);
readln
end.
```

خرج البرنامج هو:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
1
2
3
4
5
6
```

يأخذ الإجراء السابق متحولاً واحداً و هو العدد الذي سوف يقوم الإجراء بالعد إليه.عند استدعاء هذا الإجراء يُعين المتحول ضمن قوسين بعد اسم الإجراء ففي المثال السابق كانت قيمة المتحول هي على الترتيب 30 و nrtimes و الذي يساوي إلى 6 .

عندما نُعرف إجراء يحتاج إلى بعض المعلومات عند استدعائه يجب علينا أن نضيف تصريحاً عن وسطاء شكلية formal parameter ضمن القوسين الذين يأتيان بعد اسم الإجراء.
هذا التصريح عن الوسطاء يُحدد يحدد اسم ونوع كل منفذ خاص بمرور المعلومات من و إلى الإجراء أو إلى الإجراء.

تعدد الوسطاء Multiple parameters:

يمكن التصريح عن عدة وسطاء ضمن الإجراء الواحد و يمكن أن تكون هذه الوسطاء من أنواع واحدة أو أنواع مختلفة
.يحتوي السرد التالي على عدة رؤوس إجراءات مُبيناً الصيغ المختلفة لصيغ التصريح عن الوسطاء:

```
procedure countvals(howhigh:integer);
procedure bubblesort(var info:string80Array;size:integer);
procedure getinteger(message:string;var var value:integer);
procedure getstring(message:string; var value:string);
procedure getdate(var month,date,year:integer);
```

و الصيغة الكتابية لوسيط واحد و الذي يمكن أن يستخدم للتصريح عن الوسطاء اللازمة للإجراء يتكون على الشكل التالي:

{"var"}<مميز>":"<نوع المعطيات>

لاحظ أن رأس الإجراء countvals في اللائحة السابقة هو مثال واضح عن هذه الصيغة .
القوسان الكبيران حول المميز المحجوز var في الصيغة الكتابية السابقة يدلان على إمكانية استخدام هذا المميز المحجوز قبل التصريح عن الوسيط أو عدم استخدامه و هذا المميز المحجوز var سوف نرى أهميته لاحقاً.
فإذا كان لدينا عدة وسطاء تملك النوع نفسه فإننا نستطيع دمج تصريح هذه الوسطاء في تصريح واحد مع ملاحظة الفصل بين الوسطاء بفاصلة (,) كما هو واضح في رأس الإجراء getdate أما إذا كان لدينا عدة وسطاء و لكن لكل منها نوعه الخاص فيجب إيجاد تعريف خاص لكل وسيط مع ملاحظة الفصل بين تعريف الوسطاء بفاصلة منقوطة (;) كما هو مبين في رأس الإجراءين getstring,bubblesort.

الوسطاء الشكلية و المتحولات الوسيطية formal parameters and arguments :
عندما نُعرف وسيطاً شكلياً في رأس الإجراء فإننا نكون قد عرفنا منفذاً لعبور المعلومات إلى الإجراء لاستخدامها فيه وعند استدعائنا للإجراء علينا وضع هذه المعلومات على المنافذ بواسطة تحديدها أو تمريرها كمتحولات وسيطية.
ففي حين أن الوسطاء الشكلية تمثل المنافذ فإن المتحولات الوسيطية تملك المعلومات الفعلية المستخدمة ضمن الإجراء و في الحقيقة تُعرف المتحولات الوسيطية على أنها وسطاء فعلية actual parameters.
فمن أجل كل وسيط عرفناه وجب علينا تمرير متحول وسيطي إليه عند استدعاء الإجراءلذلك علينا تعيين متحولين وسيطين عند استدعائنا للإجراء getinteger في حين ينبغي علينا تعيين متحول واحد عندما نريد استدعاء الإجراء countvals.
مثلا لنأخذ رأس الإجراء التالي:

```
procedure getinteger(message:string; var value:integer);
```

وسطاء شكلية

أما استدعاء الإجراء ضمن البرنامج الرئيسي فيتم على الشكل التالي:

```
getinteger(mystr,myint);
```

وسطاء فعلية

وعليه يجري استبدال هذه المتحولات كما يلي:

formal parameters الوسطاء الشكلية	message ↑	value ↑
actual parameters الوسطاء الفعلية	mystr	myint

بالإضافة إلى ذلك يجب الانتباه إلى وضع المتحولات الوسيطية بالترتيب الصحيح فكل وسيط شكلي يملك نوعاً خاصاً به و تُعرف الوسطاء الشكلية وفق ترتيب معين لذلك يجب أن يملك كل وسيط فعلي نوعاً خاصاً به أيضاً و مطابق لنوع الوسيط الشكلي المناظر له و يجب أن يأتي بنفس الترتيب الذي أتى به الوسيط الشكلي المناظر.

وبالتأكيد فإن بعض الروتينات (الإجراءات)مسبقة التعريف مستثناة من هذه القيود فمثلاً:يمكن أن يأخذ الإجراء writeln عدداً متغيراً من المتحولات الوسيطية (الوسطاء الفعلية) و أن تكون هذه المتحولات من أنواع مختلفة عند كل استدعاء لهذا الإجراء.

سنورد الآن بعض الأمثلة على استدعاءات صحيحة و أخرى غير صحيحة معتمدين على التصريحات عن رؤوس الإجراءات في السرد السابق. افترضنا=تخيلنا في هذه الاستدعاءات أن myint متحول من النوع الصحيح و mystr متحول من النوع سلسلة رمزية و اعتماداً على ذلك فإن الاستدعاءات الخمسة التالية صحيحة و الأربعة الباقية خاطئة:

الاستدعاءات الصحيحة:

```
Countvals(55);                    (*يمكن أن تُمرر قيمة صحيحة*)
countvals(myint);                 (*يمكن للمتحول أن يُمرر على أنه وسيط ما *)
countvals(2*15);                  (*يمكن أن تمرر عبارة صحيحة*)
getinteger(mystr,myint);          (*يمكن لمتحول سلسلة رمزية أن يُمرر *)
getinteger('value?',myint);       (*يمكن لسلسلة رمزية أن تٌمرر*)
```

الاستدعاءات الخاطئة :

```
Countvals(myint,55);              (*هناك زيادة في عدد المتحولات الممررة *)
Countvals(20.3);                  (*يجب أن تكون قيمة هذا امتحول صحيحة*)
getinteger(myint);                (*المتحول الأول غير موجود: نقص في عدد المتحولات*)
getinteger(myint,'value ?');      (*هناك خطأ في ترتيب المتحولات*)
```

التصريح عن الوسطاء واستخدامها declaring and using parameters:

السؤال الذي يُطرح ما الذي يحدث عندما نصرح عن وسيط في الإجراء؟

<u>الجواب:</u>

التصريح هو عبارة عن تعليمة وظيفتها تخصيص مكان تخزين للمتحول عند استدعاء الإجراء ومكان تخزين هذا سوف يكون بالحجم الكافي لتمثيل قيمة المتحول وحسب نوع المعرف في رأس الإجراء.

فعلى سبيل المثال:عندما صرحنا عن الوسيط الشكلي howhigh على أنه من النوع الصحيح integer في الإجراء countvals فإن مكان تخزين مؤقت بطول بايتين سوف يُخصص لهذا المتحول عندما يُستدعى هذا الإجراء. و عند الانتهاء من تنفيذ الإجراء فإن هذين البايتين سيتحرران و يصبحان قابلين للاستخدامات المختلفة للذاكرة.

و السؤال الذي يطرح نفسه ما الذي يحدث عندما نمرر متحولاً وسيطياً إلى الإجراء عند استدعائه؟؟

عندما استدعينا الإجراء countvals مع القيمة (28) قام البرنامج بنسخ هذه القيمة إلى التخزين المؤقت المخصص للمتحول howhigh كما في الشكل التالي:

الوسطاء الشكلية		مواقع الذاكرة		الوسطاء الفعلية
howhigh	--------------------	28	------------------	28

و بشكل مشابه عندما استدعى البرنامج الإجراء countvals مع المتحول nrtimes و لتكن قيمته 100 مثلاً ينسخ البرنامج قيمة هذا المتحول إلى مكان التخزين المؤقت المخصص للوسيط howhigh وعندما نُسخت قيمة المتحول nrtimes لم يطرأ أي تغيير عليها و قيمتها لم تتغير كنتيجة لاستدعاء الإجراء كما في الشكل التالي:

الوسطاء الشكلية		مواقع الذاكرة			الوسطاء الفعلية
howhigh	-------------	100	100	-------------	100

عند استدعائنا لإجراء مثل countvals مع متحولات وسيطية فإن قيم هذه المتحولات فقط هي التي تهمنا و لا يهمنا مصدر هذه القيم ويُطلق على هذا التمرير بالتمرير بواسطة القيمة passed by value و بما أن قيمة هذه المتحولات تُنسخ إلى البيئة الموضعية للإجراء وجب علينا التأكد من هذه المتحولات تملك قيماً هذا يعني أنها على الأقل قد هيئت أو بمعنى آخر قد خصصت لها قيماً ابتدائية و إلا فإن الإجراء سوف يعمل بفوضى ويعطينا نتائج غير صحيحة.

يمكن أن نمرر المعلومات بواسطة قيمها بأشكال مختلفة هي:

1. على شكل قيمة ثابتة. مثال: 30 أو A string constant.
2. على شكل متحول. مثال: natimes أو GreetingStr.
3. على شكل تعبير. مثال: 3+5 أو 2<55 ....الخ

عندما نُصرح عن وسيط فإن حيز تخزين مؤقت يُخصص لهذا الوسيط و يُربط اسم الوسيطبحيز التخزين المؤقت هذا و هذا الاسم يُحدد من خلال بيئة الإجراء procedures environments و بعبارة أدق من خلال لائحة الوسطاء الملحقة باسم الإجراء. و تعرف هذه الأسماء على أنها أسماء موضعية لذلك عندما ننفذ الإجراء getinteger فإن مساحة من الذاكرة خصصت من أجل تعريف وسطائه message و value و هذه الأسماء لن تكون معروفة خارج بيئة هذا الإجراء.و بشكل مشابه فإن مساحة التخزين لعد صحيح integer قد خُصصت ضمن البيئة الموضعية local environment للإجراء countval من أجل وسيطه howhigh و ذلك عند تنفيذ هذا الإجراء.
عند الانتهاء من تنفيذ أي إجراء فإن وسطاءه الموضعية و أماكن تخزينهم تتلاشى و لا يستطيع البرنامج الرئيسي التعرف عليها.
لذلك نقول: إن استدعاء البيئة لا يعرف الأسماء الموضعية (المحلية) للوسطاء و لكنه يعرف فقط عدد و نوع المنافذ التي سوف تمرر المعلومات من خلالها إلى الإجراء.
فمثلاً عندما يستدعى الإجراء getinteger فإن أي قيمة توضع على المنفذ الأول ستعطى للمتحول message و أي قيمة توضع على المنفذ الثاني ستعطى للمتحول value.
إن المعلومات الموضوعة على المنافذ عند استدعاء البيئة يكون لها أسماء تختلف عن الأسماء المخصصة لها ضمن البيئة .
مثلاً: في البرنامج الرئيسي من الممكن أن نمرر للإجراء getinteger متحول سلسلة رمزية مثل strvariable أو ثابتاً في منفذه الأول و أن نمرر له متحولاً صحيحاً مثل myint أو قيمة محددة في منفذه الثاني.

الوسطاء ذوو الاتجاهين two way parameters :
لقد استخدمنا الوسطاء من أجل تمرير المعلومات إلى الإجراءات لتستخدمها في القيام بمهمتها (استدعاء بالقيمة) حيث أن هذه المعلومات الممررة لا تتغير ضمن الروتين المستدعى. في حين نحتاج في بعض الأحيان إلى أن نعيد معلومات من الإجراء إلى الروتين ال1ي استدعى الإجراء.فعلى سبيل المثال افترض أن الإجراء getinteger يريد أن يزويد الروتين الذي استدعاه بقيمة صحيحة إذاً يجب أن يعيد الإجراء getinteger القيمة المطلوبة عن طريق وسيطه الثاني و يبدأ بالمميز المحجوز var.
يُستخدم الوسيط الذي يبدأ التصريح عنه بواسطة var في إعادة المعلومات من الإجراء إلى الروتين المستدعى فعند الانتهاء من تنفيذ الإجراء تكون الوسطاء قد امتلكت قيماً جديدة تختلف عن القيم التي مُررت بها إلى الإجراء.
السبب في ذلك أن المتحولات الوسيطية argument التي مررت إلى الإجراء و كذلك الوسطاء الشكلية formal parameter تشير إلى مكان واحد ضمن الذاكرة.
و المثال التالي يبين لنا طريقة تمرير الوسطاء ذو الاتجاهين إذ يحتوي هذا المثال على نسختين من إجراء واحد ووظيفة هذا الإجراء هو تبديل قيمتي متحوليه الوسيطين :

Code
```
program test;
var val1,val2:integer;
procedure badintswap(first,second:integer);
var temp:integer;
begin
temp:=first;
first:=second;
second:=temp;
end;
procedure goodintswap(var first:integer; var second:integer);
var temp:integer;
begin
temp:=first;
first:=second;
second:=temp;
end;
procedure getinteger(message:string; var value:integer);
begin
```

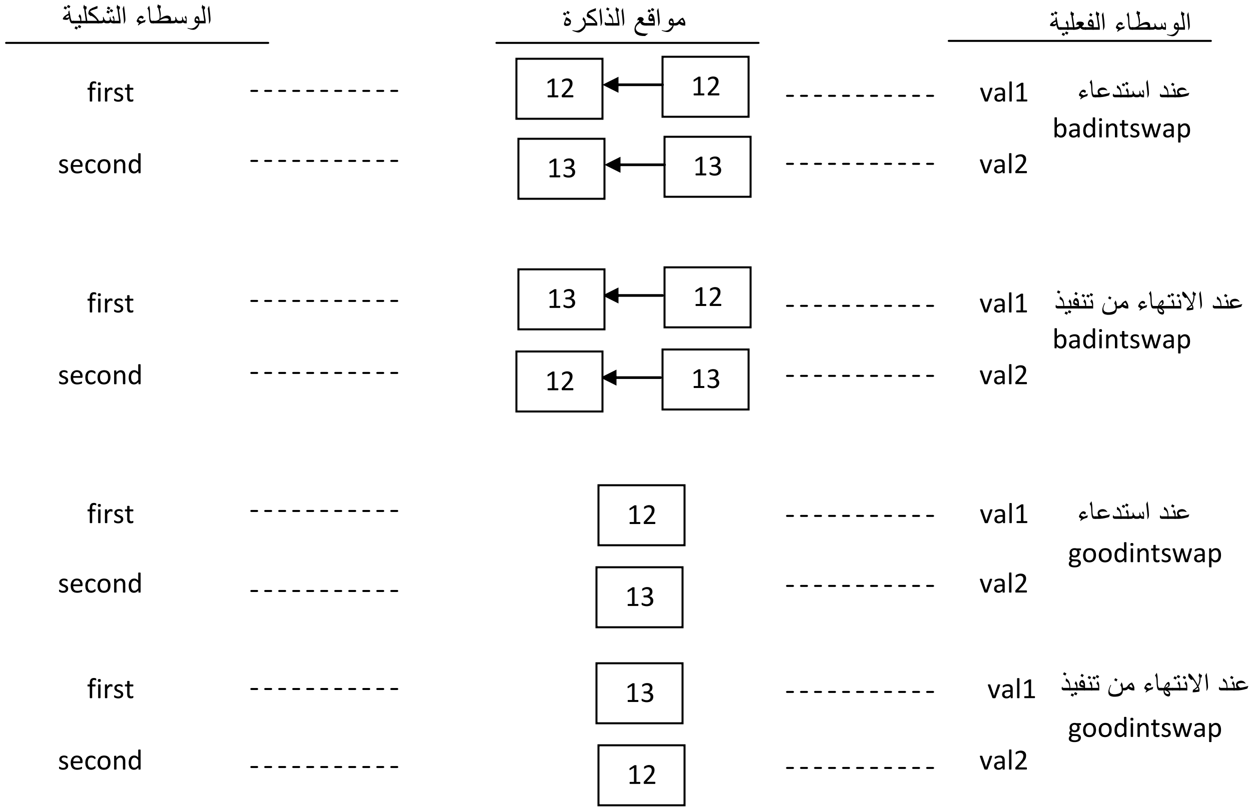
```
write(message,' ');
readln(value);
end;
begin (*main program*)
getinteger('value 1?',val1);
getinteger('value 2?',val2);
writeln('original val1 = ',val1:5,';  val2 = ',val2:5);
badintswap(val1,val2);
writeln('After badintswap');
writeln('        val1 = ',val1:5,';  val2 = ',val2:5);
goodintswap(val1,val2);
writeln('After goodintswap');
writeln('        val1 = ',val1:5,';  val2 = ',val2:5);
readln
end.
```

خرج البرنامج السابق من أجل قيم للدخل val2=13 and val1=12 كالتالي:

```
value 1? 12
value 2? 13
original  val1  =   12;  val2 =   13
After badintswap
              val1  =   12;  val2 =   13
After goodintswap
              val1  =   13;  val2 =   12
```

نلاحظ من الخرج السابق أن عملية تبديل القيم لم تحدث ضمن الإجراء badintswap في حين أن القيمتين قد بُدلتا ضمن الإجراء goodintswap و السبب في ذلك يرجع إلى التغيير الموجود في لائحة الوسطاء لكلا الإجراءين السابقين إذ سبقت وسطاء الإجراء goodintswap بالمميز المحجوز var.

يمكن توضيح ذلك من خلال الشكل المنطقي التالي:



لقد لاحظنا سابقاً عدم حصول أي عملية تبديل في قيمة المتحولين val1 و val2 بعد أن مُررت قيمها إلى الإجراء badintswap و السبب في ذلك أنه عندما مُررت هذه القيم إلى الوسطاء first و second تم تبديل قيم هذه الوسطاء و لكن هذه الوسطاء معرفة فقط ضمن الإجراء badintswap و البرنامج الرئيسي لم يتعرف على التغيير الذي حصل لهذه القيم و بالتالي لم تتغير قيمة المتحولين val1 و val2 ضمن البرنامج الرئيسي main program قبل و بعد تنفيذ الإجراء badintswap و في المقابل عندما انتهى تنفيذ الإجراء goodintswap فإن قيمة المتحولين val1 و val2 قد بُدلتا ضمن البرنامج الرئيسي و بعبارة أخرى إن الإجراء goodintswap قد علم أين توضعت قيم متحولاته الوسيطية arguments في الذاكرة و استطاع أن يُبدل قيمها. إذاً استطاع الإجراء goodintswap الوصول إلى المتحولين val1 و val2 على الرغم من أن هذه الأسماء ليست مذكورة في جسم إجراءاته و السبب في ذلك أن مواضع التخزين المخصصة للمتحولين first و second هي نفسها مواضع التخزين المخصصة للمتحولين val1 و val2 على الترتيب و هذا يعني أن التغيير الذي حصل في قيمة المتحولين first و second قد غير أيضاً المتحولين val1 و val2 .
و اعتماداً على التغيير الذي حصل بعد انتهاء الإجراء goodintswap يمكننا اعتبار أن المتحولين val1 و val2 قد مررا مباشرة و لم تمرر نسخة عن قيمتها يدعى مثل هذا النوع من التمرير للوسطاء بالتمرير المرجعي
passed by reference أو التمرير بالإشارة و الوسطاء التي تمرر بهذه الطريقة يمكن أن تتغير قيمتها عند استدعاء الإجراء. وبما أن الوسطاء الممررة مرجعياً يجب أن ترتبط بواضع تخزين في الذاكرة (حيث ستحدث التغييرات) لذلك لا يمكننا تمرير قيم مثل: (55) أو تعبير مثل (45+55) كوسطاء مرجعيين و لكن يمكننا فقط تمرير المتحولات مرجعياً حيث سيعرف الروتين المستدعى عنوان هذه المتحولات في الذاكرة و من ثم يقوم بتغيير محتواها حسب تسلسل تعليمات الروتين.

أمثلة على تمرير الوسطاء Examples Of Parameters Passing :

Code

```
program test;
var g1,g2:integer;
procedure allvalues(x,y:integer);
begin
writeln('    entering allvalues  :x = ',x,';  y= ',y);
x:=x*2;
y:=y*4;
writeln('    leaving  allvalues  :x = ',x,';  y= ',y);
end;
procedure valuevar(x:integer; var y:integer);
begin
writeln('    entering allvalues  :x = ',x,';  y= ',y);
x:=x*2;
y:=y*4;

writeln('    leaving  allvalues  :x = ',x,';  y= ',y);
end;
procedure allvars(var x,y:integer);
begin
writeln('    entering allvalues  :x = ',x,';  y= ',y);
x:=x*2;
y:=y*4;
writeln('    leaving  allvalues  :x = ',x,';  y= ',y);
end;
begin {main program }
g1:=10; g2:=20;
writeln('calling allvalues       :g1= ',g1,'; g2= ',g2);
allvalues(g1,g2);
writeln('returning from valuevar  :g1= ',g1,'; g2= ',g2);
writeln;
writeln('calling valuevar        :g1= ',g1,'; g2= ',g2);
valuevar(g1,g2);
writeln('returning from valuevar  :g1= ',g1,'; g2= ',g2);
writeln;
writeln('calling allvars         :g1= ',g1,'; g2= ',g2);
allvars(g1,g2);
writeln('returning from allvars   :g1= ',g1,'; g2= ',g2);
readln;
end.
```

أن x و y هي أسماء موضعية للمتحولين g1 و g2 على الترتيب.

خرج البرنامج السابق كالتالي :

```
calling allvalues          :g1= 10; g2= 20
     entering allvalues    :x = 10;  y= 20
     leaving  allvalues    :x = 20;  y= 80
returning from valuevar    :g1= 10; g2= 20

calling valuevar           :g1= 10; g2= 20
     entering allvalues    :x = 10;  y= 20
     leaving  allvalues    :x = 20;  y= 80
returning from valuevar    :g1= 10; g2= 80

calling allvars            :g1= 10; g2= 80
     entering allvalues    :x = 10;  y= 80
     leaving  allvalues    :x = 20;  y= 320
returning from allvars     :g1= 20; g2= 320
```


يستدعي البرنامج السابق ثلاثة إجراءات لوائح و سطائها متماثلة تقريباً إذ يملك كل إجراء وسيطين هما x و y و عدداً مختلفاً من المتحولات المسبوقة بالمميز المحجوز var فمثلاً:لا يملك الإجراء allvalues أي مميز var قبل وسطائه و هذا يعني أن متحولاته سوف تمرر بواسطة قيمها و بتعبير آخر فإن نسخة من المتحولات الوسيطية سوف تمرر إلى هذا الإجراء و أي تغيير سوف يطرأ على هذه النسخة لن يؤثر على المتحولات الأصلية (الفعلية).

أما الإجراء valuevar فإن وسيطه الأول سيمرر بواسطة قيمته ووسيطه الثاني سيمرر مرجعياً و هذا الوسيط مرتبط بالذاكرة و يستطيع الروتين المستدعى الوصول إليه و أي تغيير سوف يحصل في موقع الذاكرة هذا سوف يبقى بعد الانتهاء من تنفيذ الإجراء valuevar .

أما الإجراء allvars فإن وسيطيه سيمرران مرجعياً و هذا يعني أن كلا الوسيطين سوف يعيدان قيماً إلى الروتين المستدعي.

التوابع Functions:

لقد صممت الإجراءات للقيام بعمل ما و بالمقابل صممت التوابع لحساب قيمة ما و الإجراءات يمكن أن تعيد نتائج عملها أو لا تعيد أما التوابع فهي تعيد دائما ناتج عملها ففي علم الجبر فإن التوابع أو الدوال تنجز حسابات على قيم ابتدائية معينة أو متحولات وسيطية و تعيد دائماً قيمة واحدة . لنأخذ مثلاً التابع التالي:

$y=x^2$

فإن من أجل أي قيمة لـ x فأن التابع يعيد لنا مربع القيمة x .

مثال: x=20 فإن التابع يعيد y=400

افترض=تخيل أنه لدينا تابعاً اسمه SQRT لحساب الجذر التربيعي لقيمة تمرر له فعند استدعاء هذا التابع مع متحول وسيطي argument سيعيد هذا التابع قيمة واحدة و هي الجذر التربيعي للمتحول الممرر.

مثال:

SQRT(81)=9

SQRT(6.25)=2.5

و في لغة باسكال التوابع هي روتينات تعيد دائماً قيمة محددة. و التوابع تملك نفس مكونات الإجراءات من رأس للتابع و قسم تصريح و جسم للتابع و الصيغة الكتابية لتعريف التابع و الصيغة الكتابية لاستدعاء التابع تختلفان عن مقابلاتها بالنسبة للإجراءات . أما القواعد و القوانين المطبقة على الوسطاء فلا تختلف أبداً بين الإجراءات و التوابع

تعريف التوابع في لغة تربو باسكال Defining Functions in Turbo Pascal:

يبين النابعان التاليان الصيغة الكتابية لتعريف التوابع:

code

```
program test;
function max(one,two:real):real;
begin
if(one>two)then
max:=one
else
max:=two;
end;
function min(one,two:real):real;
begin
if(one>two)then
min:=two
else
min:=one
end;
var x,y:integer;
begin
writeln('enter two number>>>...');
readln(x,y);
writeln('max(',x,',',y,')=',max(x,y):6:2);
writeln('min(',x,',',y,')=',min(x,y):6:2);
readln
end.
```

خرج البرنامج هو:

```
enter two number>>>...
100
300
max(100,300)=300.00
min(100,300)=100.00
```


رأس التابع مشابه تماماً لرأس الإجراء مع بعض الاختلافات البسيطة و هذه الاختلافات هي:

1. يبدأ رأس التابع بالمميز المحجوز function في حين أن رأس الإجراء يبدأ بالمميز المحجوز procedure .
2. بعد أن يتبع هذا المميز المحجوز باسم للروتين و بلائحة وسطاء اختيارية ينتهي رأس التابع بنوع معطيات يُعطى لاسم التابع و في مثالنا السابق كان real .

يمكن أن يعيد التابع أي قيمة من أي نوع معطيات يحدد بعد النقطتان في رأس التابع و نوع المعطيات هذا يمكن أن يكون أي نوع معطيات بسيط بالإضافة إلى نوع معطيات سلسلة رمزية و بعض أنواع المعطيات المركبة .

يختلف جسم التابع أيضاً عن جسم الإجراء في شيء واحد فقط هو أن استدعاء التابع يعيد قيمة هذا التابع لذلك يجب أن تعطى هذه القيمة إلى التابع من خلال عبارة الإلحاق في أي مكان ضمن جسم التابع و عبارة الإلحاق هذه تجعل التابع تساوي قيمة

ما أو يساوي قيمة أحد المتحولات أو يساوي ناتج تعبير معين و لكن بشرط أن تكون هذه القيمة من نفس نوع المعطيات المخصص للتابع إذ يوضع اسم التابع على يسار معامل الإلحاق دائماً و توضع القيمة على يمين هذا المعامل.
فمثلاً: يعيد التابع السابق max قيمة للروتين المستدعي (أي الذي يستدعيه) بعد أن ينفذ إحدى عبارتي الإلحاق الموجودة ضمن جسم هذا التابع.

استدعاء التابع calling a function:

عند استدعائنا للإجراءات نكتفي بذكر اسم الإجراء مع لائحة مناسبة من المتحولات الوسيطية (في حال وجود متحولات وسيطية) أما استدعاء التابع فله طريقة مختلفة تماماً و لفهم آلية استدعاء التابع يجب أن ننوه أن التابع نظرياً عبارة عن طريقة لتوليد قيمة محددة هي قيمة التابع و في الحقيقة يُعتبر استدعاء التابع في لغة باسكال عبارة عن قيمة لذلك يمكن أن يتم استدعاء التابع في أي مكان يمكن أن توجد فيه قيمة هذا التابع و البرنامج التالي يبين لنا عدة مواضع يمكن أن يتواجد فيها استدعاء التابع:

Code

```
program test;
smaller,larger,val1,val2:real;
function max(one,two:real):real;
begin
if(one>two)then
max:=one
else
max:=two;
end;
function min(one,two:real):real;
begin
if(one>two)then
min:=two
else
min:=one
end;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin(* mian program*)
getreal('value 1?',val1);
getreal('value 2?',val2);
smaller:=min(val1,val2);
larger:=max(val1,val2);
writeln('--------------------------------------------');
writeln('val1    = ',val1:6:2,'; val2      = ',val2:6:2);
writeln('smaller = ',smaller:6:2,'; larger    = ',larger:6:2);
writeln('--------------------------------------------');
writeln('min     = ',min(val1,val2):6:2,'; max       = ',max(val1,val2):6:2);
writeln('sum     = ',min(val1,val2)+max(val1,val2):6:2);
readln
end.
```

خرج البرنامج بفرض val1=247 and val2=101 كالتالي:

```
value 1? 247
value 2? 101
---------------------------------------------
val1    = 247.00; val2      = 101.00
smaller = 101.00; larger    = 247.00
---------------------------------------------
min     = 101.00; max       = 247.00
sum     = 348.00
```

المهندس خالد الشيخ

يمكن أن يوضع استدعاء التابع في الطرف الأيمن من عبارة الإلحاق أو أن يُمرر استدعاء التابع كمتحول وسيطي للإجراءات و للتوابع الأخرى.
عندما ينفذ البرنامج و يصل التنفيذ إلى عبارة استدعاء فإنه:

1. يتوقف تنفيذ الروتين المستدعي بشكل مؤقت و ينقل التحكم إلى التابع حيث يبدأ التابع بدوره بتنفيذ تعليماته.
2. و تنفيذ التابع هذا يتم وفق تسلسل التعليمات إذ يبدأ بتنفيذ أول تعليمة في جسم التابع و يستمر التنفيذ هذا حتى تحسب قيمة مناسبة و تلحق بالتابع.
3. بعد أن تلحق هذه القيمة بالتابع تُعاد هذه القيمة هذه القيمة إلى الروتين المستدعي و الذي يستأنف تنفيذ تعليماته وفق جديدة حصل عليها من تنفيذ التابع.

بما أن التابع هو ببساطة عبارة عن قيمة فإن استدعاء التابع لا يمكن أن يظهر كعبارة مستقلة بذاتها لذلك كان استدعاء التابع لا يملك نفس شكل استدعاء الإجراء فعلى سبيل المثال : افترض=تخيل أننا استدعينا التابع التالي ضمن عبارة مستقلة كما يلي:

```
min(55,100);          (*a fancy  way of saying 55*)
55;                   (*a value is not a statement*)
```

سوف يعطينا استدعاء التابع السابق قيمة و بما أن القيمة بحد ذاتها ليست عبارة فإن استدعاء التابع و الذي يعتبر قيمة لا يمكن أن يكون عبارة.

تمرير الوسطاء إلى التوابع parameters to functions :
تُمرر الوسطاء إلى التوابع بنفس الطريقة التي بها الوسطاء إلى الإجراءات و هذا يعني أننا نستطيع تمرير الوسطاء إلى التوابع بقيمتها pass by value أو تمريرها مرجعياً (بالعنوان) pass by reference و بما أن التوابع تعيد قيمة مباشرة لذا سيكون احتياج التوابع إلى تغيير قيمة المتحولات الوسيطية أقل من حاجة الإجراءات لذلك في حين أننا بحاجة إلى تمرير هذه المتحولات بواسطة قيمتها إلى التوابع.

الإجراءات و التوابع مسبقة التعريف predefined procedure and functions:
تملك لغة باسكال كماً هائلاً من الروتينات مسبقة التعريف الممكن استخدامها ضمن برامجنا و جاءت لغة تربو باسكال بكميات إضافية من الروتينات مسبقة التعريف و أضافتها إلى هذه المجموعة من الروتينات.

أولاً: الروتينات مسبقة التعريف في لغة باسكال القياسية Predefined Routines In Standard Pascal:

```
Function ABS(intval:shortint):shortint;
Function ABS(intval:integer):integer;
Function ABS(intval:longint):longint;
Function ABS(Rval:real):real;
```

تعيد هذه الروتينات القيمة المطلقة absolute value لمتحولاتها الوسيطية و التي يجب أن تحمل قيمة مؤشرة (تملك إشارة إما + أو -) أو قيمة حقيقية و تكون القيمة المعادة من نفس نوع المعطيات التي ينتمي إليه المتحول الوسيطي. و بشكل عام القيمة المطلقة لعدد هو الشكل الموجب لهذا العدد إذاً العبارتان التاليتان تعطيان نفس الناتج:

```
Result1:=ABS(-55.0);     (* returns  55.0*)
Result2:=ABS(55.0);      (* returns  55.0*)
```

يشرح البرنامج التالي التابع ABS:

Code
program test; var val,result:real; procedure getreal(message:string;var value:real); begin write(message,' '); readln(value); end; begin repeat getreal('value (> 500 to stop)? ',val); result:=ABS(val); writeln('ABS (',val:8:3,') = ',result:8:3); until val>500.0; readln end.

خرج البرنامج أعلاه مشابه للتالي:

```
value (> 500 to stop)?       6
ABS (    6.000 ) =    6.000
value (> 500 to stop)?      -6
ABS (   -6.000 ) =    6.000
value (> 500 to stop)?    -600
ABS ( -600.000 ) =  600.000
value (> 500 to stop)?     400
ABS (  400.000 ) =  400.000
value (> 500 to stop)?    -500
ABS ( -500.000 ) =  500.000
value (> 500 to stop)?     501
ABS (  501.000 ) =  501.000
```

لدينا الروتين التالي:

Function ARCTAN(value:real):real;

يعيد هذا الروتين قيمة مثلثية trigonometric value عبارة عن قيمة الزاوية التي ظلها tangent ممرر ضمن المتحول الوسيطي الذي يجب أن تكون قيمته حقيقية و سوف يعيد هذا التابع قيمة حقيقية أيضاً و البرنامج التالي يشرح لنا التابع ARCTAN :

Code
program test; var val,result:real; procedure getreal(message:string;var value:real); begin write(message,' '); readln(value); end; begin repeat getreal('value (> 10.0 to stop)? ',val);

```
result:=ARCTAN(val);
writeln('ARCTan ( ',val:8:3,' ) = ',result:8:3);
until val>10.0;
readln
end.
```

```
value (> 10.0 to stop)?       0
ARCTan (    0.000 ) =    0.000
value (> 10.0 to stop)?       1
ARCTan (    1.000 ) =    0.785
value (> 10.0 to stop)?      -1
ARCTan (   -1.000 ) =   -0.785
value (> 10.0 to stop)?       2
ARCTan (    2.000 ) =    1.107
value (> 10.0 to stop)?       3
ARCTan (    3.000 ) =    1.249
value (> 10.0 to stop)?      -3
ARCTan (   -3.000 ) =   -1.249
value (> 10.0 to stop)?       7
ARCTan (    7.000 ) =    1.429
value (> 10.0 to stop)?      10
ARCTan (   10.000 ) =    1.471
value (> 10.0 to stop)?      12
ARCTan (   12.000 ) =    1.488
```

```
Function CHR(ordvalue:BYTE):CHAR;
```

يعيد هذا التابع الرمز الذي حُددت قيمة ترتيبه (أي شفرة ASCII) لهذا الرمز.
و البرنامج التالي يوضح عمل الإجراء CHR:

Code

```
program test;
const carriagereturn=13;
backspace=8;
var result:integer;
begin
result:=2500;
write(result:30);
write(chr(backspace),5);
write(chr(carriagereturn));
write(result:20);
write(chr(backspace),2);
write(chr(carriagereturn));
write(result:10);
write(chr(backspace),3);
write(chr(carriagereturn));
writeln(Chr(97));
readln
end.
```

خرج البرنامج السابق هو:

```
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
a      2503      2502      2505
```

<u>أنجز البرنامج السابق مجموعة من الأعمال :</u>

- كتب قيمة المتحول result و هي 2500 وفق عرض كبير للمجال field width و لكن بدون أن ينقل المؤشر إلى بداية سطر جديد.وعرض المجال هذا أخذ بالتناقص في كل مرة كتبت فيها قيمة المتحول result ضمن البرنامج.
- كتب رمز الحذف الخلفي back space و الذي شفرة ASCII المقابلة له هي 8 و هذا الرمز ينقل المشيرة cursor عموداً واحداً إلى الخلف فوق آخر رقم من أرقام العدد 2500 .
- كتب رقماً جديداً عوضاً عن الرقم الأخير في العدد 2500 الذي حذفه.
- كتب رمز الإرجاع carriage return و الذي شفرة ASCII المقابلة له هي 13 و هذا الرمز يُرجع المشيرة إلى أول عمود ضمن السطر المحدد (أي السطر الموجودة فيه المشيرة) و حتى ننتقل إلى سطر جديد نحن بحاجة إلى رمز التغذية السطرية linefeed و الذي يتقل المشيرة إلى بداية سطر جديد.

Function cos(radians :real):real;

يُرجع هذا الروتين تجيب cosine الزاوية المحددة بالمتحول الحقيقي الممرر إلى التابع و قيمة هذا التابع ستكون ضمن المجال من +1.0 إلى -1.0. أما قيمة المتحول الممرر فيجب أن تكون مقدرة بالراديان radian و ليست بالدرجات degree و القياسات بالراديان هي عبارة عن توابع أو مضاريب للقيمة $\pi = 3.14159265$ فعلى سبيل المثال: القيمة 180 درجة تكافئ القيمة π راديان و هذا يعني أن 1 راديان هو حوالي 57° درجة.والبرنامج التالي يبين عمل التابع cos:

Code

```
program test;
var cosine,angle:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
repeat
getreal('Angle (in radians> 100.0 to stop)?',angle);
cosine:=cos(angle);
writeln('Cosine of ',angle:6:4,' radiane = ',cosine:6:4);
until angle>100.0;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
Angle (in radians) 100.0 to stop)? 1
Cosine of 1.0000 radiane = 0.5403
Angle (in radians) 100.0 to stop)? 0
Cosine of 0.0000 radiane = 1.0000
Angle (in radians) 100.0 to stop)? 2
Cosine of 2.0000 radiane = -0.4161
Angle (in radians) 100.0 to stop)? 10
Cosine of 10.0000 radiane = -0.8391
Angle (in radians) 100.0 to stop)? 100
Cosine of 100.0000 radiane = 0.8623
Angle (in radians) 100.0 to stop)? 101
Cosine of 101.0000 radiane = 0.8920
```

Function EXP(valtoraise:real):real;

يعيد هذا التابع ناتج رفع العدد النيبري e إلى قيمة المتحول الوسيطي الممرر لهذا التابع مع العلم بأن 2.7181628 ≡e
و البرنامج التالي يوضح عمل الروتين EXP للقيم التي ندخلها:

Code

```
program test;
var val,result:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
writeln;
repeat
getreal('value?  (>10.0 to stop)?',val);
result:=exp(val);
writeln('EXP (',val:10:3,') = ', result:10:3);
until val>10.0;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value?  (>10.0 to stop)? 0
EXP (     0.000) =      1.000
value?  (>10.0 to stop)? 1
EXP (     1.000) =      2.718
value?  (>10.0 to stop)? 2
EXP (     2.000) =      7.389
value?  (>10.0 to stop)? 10
EXP (    10.000) =  22026.466
value?  (>10.0 to stop)? 11
EXP (    11.000) =  59874.142
```

```
Function LN(val:real):real;
```

يعيد هذا التابع قيمة اللوغاريتم الطبيعي natural logarithm (اللوغاريتم الذي أساسه e) للمتحول الوسيطي الحقيقي الذي يمرر إليه و قيمة هذا المتحول هذه يجب أن تكون أكبر تماما من 0.0 .
و اللوغاريتم الطبيعي لعدد هو قوة العدد النيبري e و الذي يعطينا بعد رفع العدد e إليه العدد المطلوب حساب اللوغاريتم الطبيعي له .
فمثلاً: $e^2 \equiv 7.389056$ و هذا يعني أن $LN(7.389056) \cong 2$
و البرنامج التالي يوضح استخدام كلا من التابعين المتعاكسين LN and EXP .

Code

```
program test;
var val,result,Lresult:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
writeln;
repeat
getreal('value?  (<0 to quit)',val);
result:=exp(val);
writeln('EXP (',val:10:3,') = ', result:10:3);
Lresult:=LN(result);
writeln('LN (',result:10:3,') = ', Lresult:10:3);
until val<0.0;
readln
end.
```

خرج البرنامج شبيه بالتالي:

```
value?  (<0 to quit)     0
EXP (     0.000) =     1.000
LN (     1.000) =     0.000
value?  (<0 to quit)     1
EXP (     1.000) =     2.718
LN (     2.718) =     1.000
value?  (<0 to quit)     2
EXP (     2.000) =     7.389
LN (     7.389) =     2.000
value?  (<0 to quit)     8
EXP (     8.000) =  2980.958
LN (  2980.958) =     8.000
value?  (<0 to quit)    -1
EXP (    -1.000) =     0.368
LN (     0.368) =    -1.000
```

```
Function ODD(val:longint):Boolean;
```

يعيد هذا التابع القيمة true إذا كان العدد الممرر إلى التابع فردياً (بمعنى آخر باقي قسمته على العدد 2 هو 1) و إلا فإنه يعيد القيمة False . و المتحول الوسيطي الممرر يجب أن يكون عدداً صحيحاً.

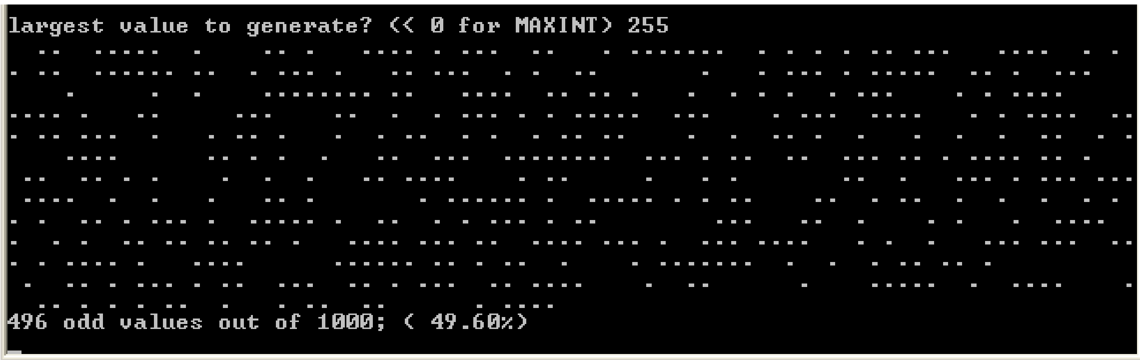
<u>و البرنامج التالي يوضح عمل التابع odd :</u>

Code

```
program test;
var count,oddcount,maxval,result:integer;
const maxtrials=1000;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
begin
oddcount:=0;
randomize;
getinteger('largest value to generate? (< 0 for MAXINT)',maxval);
if (maxval<0)then
maxval:=maxint;
for count:=1 to maxtrials do
begin
result:=random(maxval);
if(odd(result))then
begin
write('.');
inc(oddcount);
end
else
write(' ');
end;
writeln;
write(oddcount,' odd values out of ',maxtrials);
writeln('; (', (oddcount/maxtrials)*100:6:2,'%)');
readln;
end.
```

يولد البرنامج السابق قيمة عشوائية صحيحة فإذا كانت هذه القيمة فردية يقوم بطباعة نقطة و إما إذا كانت زوجية فيقوم بكتابة فراغ و يكرر هذه العملية 1000 مرة.

خرج البرنامج السابق شبيه بالخرج التالي (من أجل maxval=255):



حاول تغيير قيمة maxval إلى قيمة أكبر أو غير عدد مرات تكرار عملية تغيير القيم ضع المجال أكبر من 5000 في المتحول الثابت maxtrials أو أعد تنفيذ البرنامج أكثر من مرة ماذا نلاحظ ؟؟ !.
في الحقيقية سوف تقترب نسبة الأعداد الفردية من 50 بالمائة و السبب في ذلك أن القيم العشوائية المتولدة سوف تكون وفق أحد احتمالين إما فردية أو زوجية أي أن احتمال كون القيمة العشوائية المتولدة فردية هو 50 بالمائة نظرياً.

```
Function ORD(val:byte):longint;
Function ORD(val:shortint):longint;
Function ORD(val:integer):longint;
Function ORD(val:word):longint;
Function ORD(val:longint):longint;
Function ORD(val:Char):longint;
Function ORD(val:Boolean):longint;
```

يعيد هذا الروتين قيمة ترتيب المتحول الوسيطي الممر إليه و الذي ينتمي إلى أحد أنواع المعطيات الترتيبية ordinal type و هذا يعني أنه يمكن أن ينتمي إلى أحد أنواع المعطيات التالية:

1. أحد أنواع المعطيات الصحيحة.
2. نوع المعطيات رمز.
3. نوع المعطيات بولياني.
4. نوع المعطيات التعدادي.

فمن أجل نوع المعطيات رمز char تكون القيمة المعادة بواسطة هذا التابع هي قيمة شفرة ASCII المقابلة لقيمة المتحول الوسيطي. أما إذا كان المتحول الوسيطي من أحد أنواع الأعداد الصحيحة فإن القيمة التي يعيدها التابع هي نفس قيمة المتحول الوسيطي. و إذا كانت قيمة المتحول الوسيطي بولیانیاً فإن القيمة التي يعيدها هي إما 0 إذا كانت قيمة المتحول FALSE أو 1 إذا كانت قيمة المتحول TRUE. أما إذا كان كان المتحول من النوع التعدادي فإن قيمة هذا التابع ستساوي إلى موقع قيمة هذا المتحول ضمن اللائحة التعدادية مطروحاً منه القيمة 1 و هذا اعتماداً على أن أول عنصر في اللائحة التعدادية قيمة ترتيبه 0 و العنصر الثاني قيمة ترتيبه 1 و هكذا. و البرنامج التالي يشرح استخدام التابع ORD:

Code
program test; type mytype=(val_a,val_b,val_c); var t1:integer; c:char; b:boolean; m:mytype; begin t1:=-1;

```
c:='a';
b:=false;
m:=val_c;
writeln('ord ',t1,'=',ord(t1));
writeln('ord ',c,'=',ord(c));
writeln('ord ',b,'=',ord(b));
writeln('ord m =',ord(m));
writeln('ord ',not b,'=', ord(not(b)));
readln;
end.
```

خرج البرنامج هو:

```
ord -1=-1
ord a=97
ord FALSE=0
ord m =2
ord TRUE=1
```

المهندس خالد ياسين الشيخ

```
Function pred(val:byte):byte;
Function pred(val:shortint):shortint;
Function pred(val:integer):integer;
Function pred(val:word):word;
Function pred(val:longint):longint;
Function pred(val:char):char;
Function pred(val:Boolean):Boolean;
```

يعيد هذا الروتين القيمة السابقة لقيمة متحوله الوسيطي الذي ينتمي إلى أحد الأنواع الترتيبية التي ذكرناها آنفاً و ناتج هذا التابع سيكون غير معروف إذا استدعينا هذا التابع من أجل أصغر قيمة في هذه الأنواع المرتبة. فمثلاً إذا أردنا من التابع أن يجلب لنا القيمة السابقة لأول قيمة في النوع التعدادي فإن ناتج هذا التابع سيكون غير معرف. و البرنامج التالي يشرح لنا كيفية استخدام تابع السلف predecessor function المسمى التابع pred:

Code

```
program test;
type
mytype=(val_a,val_b,val_c);
var
t1:integer;
c:char;
b:boolean;
m:mytype;
begin
t1:=55;
c:='q';
b:=false;
m:=val_c;
writeln('pred ',t1,'=',pred(t1));
writeln('pred ',c,'=',pred(c));
writeln('pred ',b,'=',pred(b));
writeln('pre m =',ord(pred(m)));
writeln('pred ',not b,'=', pred(not(b)));
readln;
end.
```

خرج البرنامج السابق:

```
pred 55=54
pred q=p
pred FALSE=TRUE
pre m =1
pred TRUE=FALSE
```

```
Procedure read;
Procedure readln;
```

يستخدم الإجراءان read و readln لقراءة المعلومات من ملف أو من جهاز الدخل القياسي (أي لوحة المفاتيح keyboard).

```
Function ROUND(val:real):longint;
```

يُحول هذا الروتين قيمة حقيقية real إلى قيمة من النوع الصحيح الطويل longint و ذلك بواسطة تدوير القسم الكسري فإذا كانت قيمة القسم الكسري من العدد الحقيقي أقل من 0.5 فسيحذف القسم الكسري فقط و إلا فيزداد العدد الصحيح بمقدار 1 مع حذف القسم الكسري.
يكون ناتج هذا التابع في لغة باسكال من النوع الصحيح integer في حين أنه في لغة تربو باسكال يكون من النوع longint و البرنامج التالي يوضع عمل التابع round:

Code

```
program test;
var result,rval:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin (*main program*)
repeat
getreal('value (> 500 to stop)? ',rval);
result:=round(rval);
writeln('round ( ',rval:8:3, ') = ',result:8:3);
until rval>500.0;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value (> 500 to stop)?   5.65
round (     5.650) =    6.000
value (> 500 to stop)?  77.23
round (    77.230) =   77.000
value (> 500 to stop)?  77.52
round (    77.520) =   78.000
value (> 500 to stop)?  -66.6
round (   -66.600) =  -67.000
value (> 500 to stop)?  200.36
round (   200.360) =  200.000
value (> 500 to stop)?  501.00
round (   501.000) =  501.000
```

Function sin (radians:real):real;

يعيد هذا التابع جيب الزاوية sine المحددة بالمتحول الوسيطي الذي يجب أن تكون حقيقياً و يجب أن تكون قيمة الزاوية هذه مقاسة بالراديان و ليست الدرجات فالراديان عبارة عن توابع أو مضاريب للقيمة $\pi \cong 3.14159265$ حيث تكافئ القيمة 180 درجة المقدار π راديان و هذا يعني أن رادياناً واحداً يساوي تقريباً 57.2 درجة و ناتج هذا التابع سيكون ضمن المجال من 1.0+ إلى 1.0- .

ويوضح البرنامج التالي عمل التابع sin :

Code

```
program test;
var sine,angle:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin (*main program*)
repeat
getreal('Anale (in radians; >100  to stop)? ',Angle);
sine:=sin(angle);
writeln('sine of ',angle:6:4,
  ' radians = ',sine:6:4);
until angle>100;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
Anale (in radians; >100  to stop)?   5
sine of 5.0000 radians = -0.9589
Anale (in radians; >100  to stop)?   2.2
sine of 2.2000 radians = 0.8085
Anale (in radians; >100  to stop)?   3
sine of 3.0000 radians = 0.1411
Anale (in radians; >100  to stop)?   6
sine of 6.0000 radians = -0.2794
Anale (in radians; >100  to stop)?   4
sine of 4.0000 radians = -0.7568
Anale (in radians; >100  to stop)?   4.4
sine of 4.4000 radians = -0.9516
Anale (in radians; >100  to stop)?   101
sine of 101.0000 radians = 0.4520
```

```
Function SQR(val:shortint):shortint;
Function SQR(val:integer):integer;
Function SQR(val:longint):longint;
Function SQR(val:real):real;
```

يعيد هذا الروتين ناتج تربيع قيمة المتحول الوسيطي أي يعيد الناتج val*val الذي سيكون من نفس نوع المعطيات التي ينتمي إليها المتحول الوسيطي أي أما نوع صحيح أو حقيقي(أي إذا مررنا إلى هذا التابع قيمة صحيحة integer فسيكون ناتج هذا التابع قيمة صحيحة أيضاً integer) و لكن يجب الانتباه هنا إلى احتمال حدوث حالة فيض overflow عند استخدام هذا التابع مع الأعداد الصحيحة. و البرنامج التالي يبين لنا التابع SQR:

Code

```
program test;
var val,result:longint;
procedure getreal(message:string; var value:longint);
begin
write(message,' ');
readln(value);
end;
begin (*main program*)
repeat
getreal('value (<-500 to stop)? ',val);
result:=SQR(val);
writeln('SQR ( ',val:10, ') = ',result:15);
until val<-500;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value (<-500 to stop)?     5
SQR (       5) =          25
value (<-500 to stop)?    -5
SQR (      -5) =          25
value (<-500 to stop)?    20
SQR (      20) =         400
value (<-500 to stop)?   600
SQR (     600) =      360000
value (<-500 to stop)?    50
SQR (      50) =        2500
value (<-500 to stop)?   -50
SQR (     -50) =        2500
value (<-500 to stop)?  -501
SQR (    -501) =      251001
```

و البرنامج التالي يستخدم التابع SQR لتوضيح العلاقة بين تابع الجيب sine و تابع التجيب cosine:

Code

```
program test;
var sine,cosine,angle:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin (*main program*)
writeln('   sin(x) ', '   cos(x)  ',
        '      SQR(sine) + SQR(cosine)');
repeat
getreal('Angle (>100 to stop)? ',Angle);
sine:=SIN(Angle);
cosine:=cos(Angle);
writeln(sine:10:4,cosine:10:4,
            SQR(sine)+SQR(cosine):20:4);
until Angle>100;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
    sin(x)     cos(x)              SQR(sine) + SQR(cosine)
Angle (>100 to stop)?    2.00
    0.9093    -0.4161                     1.0000
Angle (>100 to stop)?    1.00
    0.8415     0.5403                     1.0000
Angle (>100 to stop)?    0.00
    0.0000     1.0000                     1.0000
Angle (>100 to stop)?    5.00
   -0.9589     0.2837                     1.0000
Angle (>100 to stop)?    6.00
   -0.2794     0.9602                     1.0000
Angle (>100 to stop)?    4.00
   -0.7568    -0.6536                     1.0000
Angle (>100 to stop)?    8.00
    0.9894    -0.1455                     1.0000
Angle (>100 to stop)?   12.00
   -0.5366     0.8439                     1.0000
Angle (>100 to stop)?  100.00
   -0.5064     0.8623                     1.0000
Angle (>100 to stop)?  101.00
    0.4520     0.8920                     1.0000
```

```
Function SQRT(val:shortint):shortint;
Function SQRT(val:integer):real;
Function SQRT(val:longint):real;
Function SQRT(val:real):real;
```

يعيد هذا التابع الجذر التربيعي لقيمة المتحول الوسيطي الممررة إليه و التي يجب أن تكون قيمة غير سالبة يمكن أن يكون هذا المتحول من أي نوع صحيح أو من النوع الحقيقي و لكن ناتج هذا التابع دائماً من النوع الحقيقي.و البرنامج التالي يوضح عمل التابع SQRT:

Code

```
program test;
var val,result:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin (*main program*)
getreal('value (<0 to stop)? ',val);
while(val>0)do
begin
result:=sqrt(val);
writeln('SQRT ( ',val:8:3,') = ',result:8:3);
getreal('value (<0 to stop)? ',val);
end;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value (<0 to stop)?   25
SQRT (   25.000) =     5.000
value (<0 to stop)?   100
SQRT (  100.000) =    10.000
value (<0 to stop)?   200
SQRT (  200.000) =    14.142
value (<0 to stop)?   2
SQRT (    2.000) =     1.414
value (<0 to stop)?   49
SQRT (   49.000) =     7.000
value (<0 to stop)?   169
SQRT (  169.000) =    13.000
value (<0 to stop)?   2500
SQRT ( 2500.000) =    50.000
value (<0 to stop)?   0
```

```
Function succ(val:byte):byte;
Function succ(val:shortint):shortint;
Function succ(val:integer):integer;
Function succ(val:word):word;
Function succ(val:longint):longint;
Function succ(val:char):char;
Function succ(val:Boolean):Boolean;
```

يعيد هذا الروتين القيمة التالية لقيمة المتحول الوسيطي الممررة إليه و الذي ينتمي إلى أحد أنواع المعطيات المرتبة التي ذكرناها سابقاً و ناتج هذا التابع لن يكون معرفاً إذا طلبنا منه أن يجلب لنا القيمة التالية لآخر قيمة من قيم نوع المعطيات الذي ينتمي إليه هذا المتحول.

والبرنامج التالي لنا عمل التابع الخلف (القيمة التالية) successor function المسمى تابع succ :

Code
```
program test;
type
mytype=(val_a,val_b,val_c);
var I:byte;
c:char;
b:boolean;
m:mytype;
begin
i:=25;
c:='d';
b:=true;
m:=val_c;
writeln('succ (',I,')= ',succ(I));
writeln('succ (',c,')= ',succ(c));
writeln('succ (',b,')= ',succ(b));
writeln('succ (',b,')= ',succ(pred(b)));
writeln('succ ( m )= ',ord(succ(m)));
writeln('succ (',not b,')= ',succ(not (b)));
readln
```

```
end.
```

خرج البرنامج السابق هو:

```
succ (25)= 26
succ (d)= e
succ (TRUE)= TRUE
succ (TRUE)= TRUE
succ ( m )= 3
succ (FALSE)= TRUE
```

```
Function TRUNC(val:real):longint;
```

يُحول هذا الروتين القيمة الحقيقية إلى قيمة صحيحة طويلة و ذلك بحذف القسم الكسري من هذه القيمة. و ناتج هذا التابع يكون من النوع longint في لغة التربو باسكال و في لغة باسكال القياسية يكون من النوع integer .
يعطينا التابعان TRUNC و ROUND إمكانية تحويل قيمة حقيقية إلى قيمة صحيحة. وللمقارنة بين سلوك التابعين TRUNC و ROUND :

Code

```
program test;
var val,result:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
repeat
getreal('value (> 500 to stop)? ',val);
result:=TRUNC(val);
writeln('TRUNC ( ',val:8:3,') = ',result:8:3);
result:=round(val);
writeln('ROUND ( ',val:8:3,') = ',result:8:3);
until val>500;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value (> 500 to stop)?    55.5
TRUNC (   55.500) =   55.000
ROUND (   55.500) =   56.000
value (> 500 to stop)?    66.2
TRUNC (   66.200) =   66.000
ROUND (   66.200) =   66.000
value (> 500 to stop)?    66.5
TRUNC (   66.500) =   66.000
ROUND (   66.500) =   67.000
value (> 500 to stop)?    88.8
TRUNC (   88.800) =   88.000
ROUND (   88.800) =   89.000
value (> 500 to stop)?    23.231
TRUNC (   23.231) =   23.000
ROUND (   23.231) =   23.000
value (> 500 to stop)?    23.621
TRUNC (   23.621) =   23.000
ROUND (   23.621) =   24.000
value (> 500 to stop)?   501.6
TRUNC (  501.600) =  501.000
ROUND (  501.600) =  502.000
```

```
Procedure write;
Procedure writeln;
```

يستخدم الإجراءان السابقان للكتابة على الشاشة أو على ملف.

الروتينات مسبقة التعريف في لغة تربو باسكال Routines predefined in turbo pascal:

```
Function concat (str1,……,strx:string):string;
```

يُنشى هذا الروتين سلسلة رمزية طويلة بعد عملية ضم لكل متحولات السلاسل الممررة إليه و يعيد بعدها سلسلة رمزية طويلة.
فعندما نمرر سلسلتين رمزيتين هما str1 و str2 إلى هذا التابع فإن ناتج هذا التابع هو سلسلة رمزية هي عبارة عن اتحاد هاتين السلسلتين إذ توضع السلسلة str1 أولاً و تليها السلسلة str2. ويمكن لهذا التابع أن يأخذ سلسلتين أو أكثر كمتحولات وسيطيه لضمها و جعلها سلسلة رمزية واحدة.
و البرنامج التالي يبين لنا عمل التابع CONCAT :

Code

```
program test;
var str1, str2, str3, str4, largstr:string;
result:integer;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
randomize;
```

```
result:=random(maxint);
getstring('str1?',str1);
getstring('str2?',str2);
getstring('str3?',str3);
getstring('str4?',str4);
if(odd(result))then
begin
writeln('concatenating str1 through str4');
largstr:=concat(str1,str2,str3,str4);
end
else
begin
writeln('concatenating str4 through str1');
largstr:=concat(str4,str3,str2,str1);
end;
writeln(largstr);
readln
end.
```

يطلب البرنامج السابق من المستخدم user أربع سلاسل رمزية و بعد أن يقرأ هذه السلاسل يولد قيمة عشوائية .فإذا كانت هذه القيمة فردية فإن البرنامج يضم هذه السلاسل وفق ترتيبها الصحيح أما إذا كانت القيمة العشوائية المتولدة زوجية فإن البرنامج يضم هذه السلاسل وفق ترتيب عكسي أي أن السلسلة الرمزية الناتجة سوف تكون ناتجة عن Str4 و str3 و str2 و str1 .

خرج البرنامج السابق قد يكون شبيه بالتالي:

```
str1?  Syria Arab Republic
str2?  syria in arabic
str3?  arabic in syria
str4?  khaled is syria arabic
concatenating str1 through str4
 Syria Arab Republic syria in arabic arabic in syria khaled is syria arabic
```

```
Function COPY(str:string;    start,Nrchars:integer):string;
```

ينسخ هذا الروتين سلسلة رمزية جزئية من السلسلة الرمزية str . يبدأ النسخ من الرمز الذي رقمه مُمرر start و طول هذه السلسلة الرمزية مُمرر ضمن المتحول nrchars.

ينسخ الاستدعاء التالي للتابع copy أول ثمانية رموز من السلسلة الرمزية str و يلحق هذه السلسلة المنسوخة بالمتحول resultstr.

Resultstr:=copy(str,1,8);

إذا وصل التابع copy أثناء قيامه بعملية النسخ إلى نهاية السلسلة str و قبل أن ينتهي من عملية النسخ فإن التابع copy سُعيد سلسلة جزئية ناقصة. فمثلاً إذا كانت السلسلة str بطول 10 رموز و طلبنا من التابع copy أن ينسخ خمسة رموز ابتداء من الرمز السابع فإن هذا التابع سيعيد الرموز الثلاثة الأخيرة فقط من السلسلة str. أما إذا كانت قيمة start (أي رقم رمز بداية النسخ) أكبر من طول السلسلة str فإن التابع copy سيعيد سلسلة فارغة.

و البرنامج التالي يبين لنا عمل التابع copy:

Code
program test; const emptystring=''; var val,result:string;   startindex, nrch:integer; procedure getinteger(message:string; var value:integer); begin write(message,' '); readln(value); end; procedure getstring(message:string; var value:string); begin write(message,' '); readln(value); end; begin {mian program} getstring('string (Enter to stop)? ',val); while(val<>emptystring) do begin getinteger('starting position? ',startindex); getinteger('# chars to copy? ',nrch); result:=copy(val,startindex,nrch); writeln('start copy is ',startindex:2,'of numbers char copy(substring) is ',   nrch:2,' = ',result); getstring('string (enter to stop)? ',val); end; end.

خرج البرنامج شبيه بالتالي:

```
string (Enter to stop)?  syria arabic
starting position?  7
# chars to copy?  6
start copy is  7of numbers char copy(substring) is  6 = arabic
string (enter to stop)?
```

المهندس خالد ياسين الشيخ

```
Procedure DEC(var valtoDec:integer);
Procedure DEC(var valtoDec:longint);
Procedure DEC(var valtoDec:char);
Procedure DEC(var valtoDec:Boolean);
```

ينقص هذا الروتين قيمة المتحول الأول الممرر إليه حسب قيمة المتحول الاختياري الثانيفإذا لم نمرر لهذا الإجراء متحولاً ثانياً فإن الإجراء ينقص قيمة المتحول الأول valtoDec بمقدار واحد. ويجب أن يكون المتحول الأول ينتمي إلى أحد أنواع المعطيات المرتبة.و البرنامج التالي يشرح لنا عمل التابع DEC:

Code

```
program test;
var I:integer;
L:longint;
C:char;
B:boolean;
procedure Disptype(Ival:integer;
            Lval:longint;
            Cval:char;
            Bval:boolean);
begin

writeln('Integer value = ',Ival);
writeln('Longint value = ',Lval);
writeln('Char value    = ',Cval);
writeln('Boolean value = ',Bval);
end;
begin
I:=30; L:=55000;
C:='g';  B:=true;
DEC(I);  DEC(L);
DEC(C);  DEC(B);
Disptype(I,L,C,B);
DEC(I,3); DEC(L,3);
DEC(C,3); DEC(B,0);
Disptype(I,L,C,B);
readln
end.
```

خرج البرنامج السابق هو:

```
Integer value = 29
Longint value = 54999
Char value    = f
Boolean value = FALSE
Integer value = 26
Longint value = 54996
Char value    = c
Boolean value = FALSE
```

المهندس خالد ياسين الشيخ

Procedure Delete (var str:string; startingindex,nrTodelete:integer);

يحذف هذا الإجراء سلسلة رمزية جزئية من السلسلة str إذ تبدأ السلسلة الجزئية من الرمز الذي رقمه موجود في startingindex و طول هذه السلسلة الجزئية موجود في nrTodelete. يبدأ الإجراء بحذف الرموز من الموقع startingindex و حتى نهاية السلسلة الجزئية فإذا كانت السلسلة الجزئية أطول من السلسلة str فإن الحذف سوف ينتهي عند نهاية السلسلة str أما إذا كانت بداية السلسلة الجزئية أكبر من السلسلة str فلن يقوم الإجراء بشيء.فمثلاً تحذف العبارة التالية أول تسعة رموز من السلسلة str:

delete(str,1,9);

و البرنامج التالي يوضح عمل الإجراء delete:

Code

```
program test;
const emptystring='';
var val:string;
startindex,nrch:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
getstring('string (Enter to stop)? ',val);
while(val<>emptystring)do
begin
getinteger('starting position? ',startindex);
getinteger('# chars to delete? ',nrch);
delete(val,startindex,nrch);
writeln('remaining string = ',val);
getstring('string (Enter to stop)? ',val);
end;
end.
```

خرج البرنامج قد يكون شبيه بالتالي:

```
string (Enter to stop)?  syria arabic
starting position?  1
# chars to delete?  6
remaining string = arabic
string (Enter to stop)?  syria arabic
starting position?  3
# chars to delete?  4
remaining string = syarabic
string (Enter to stop)?
```

```
Function Frac(val:real):real;
```

يعيد هذا التابع القسم الكسري من العدد val فالعدد الممر إلى هذا النابع و ناتج هذا التابع هما عددان حقيقيان.إذا كانت قيمة المتحول myreal هي (55.454566) عندئذ ناتج العبارة التالية result=frac(myreal); هو 0.454566 و سوف يلحق الناتج بالمتحول result و البرنامج التالي يوضح عمل التابع frac :

Code

```
program test;
const emptystring='';
var val,result:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
repeat
getreal('value (> 500 to stop)? ',val);
result:=frac(val);
writeln('FRAC ( ',val:8:3,') = ',result:8:3);
until val>500.0;
readln;
end.
```

خرج البرنامج شبيه بالتالي:

```
value (> 500 to stop)?   5.236
FRAC (    5.236) =    0.236
value (> 500 to stop)?   66.642
FRAC (   66.642) =    0.642
value (> 500 to stop)?   72.222
FRAC (   72.222) =    0.222
value (> 500 to stop)?   400.235
FRAC (  400.235) =    0.235
value (> 500 to stop)?   500.69
FRAC (  500.690) =    0.690
```

```
Procedure INC(var valtoINC:integer);
Procedure INC(var valtoINC:longint);
Procedure INC(var valtoINC:char);
Procedure INC(var valtoINC:Boolean);
```

يزيد هذا الروتين قيمة المتحول الممرر إليه حسب قيمة المتحول الثاني الاختياري الممرر إليه و في حال عدم تمرير متحول ثاني إلى هذا الإجراء فسيزيد قيمة متحوله الأول بمقدار واحد شريطة أن يكون المتحول الأول من نوع معطيات ترتيبي و البرنامج التالي يشرح لنا الإجراء INC:

Code

```
program test;
var I:integer;
L:longint;
C:char;
B:boolean;
procedure Disptype(Ival:integer;
```

```
            Lval:longint;
            Cval:char;
            Bval:boolean);
begin

writeln('Integer value = ',Ival);
writeln('Longint value = ',Lval);
writeln('Char value    = ',Cval);
writeln('Boolean value = ',Bval);
end;
begin
I:=30; L:=55000;
C:='g';  B:=true;
INC(I);  INC(L);
INC(C);  INC(B);
Disptype(I,L,C,B);
INC(I,3); INC(L,3);
INC(C,3); INC(B,0);
Disptype(I,L,C,B);
readln
end.
```

خرج البرنامج السابق هو:

```
Integer value = 31
Longint value = 55001
Char value    = h
Boolean value = TRUE
Integer value = 34
Longint value = 55004
Char value    = k
Boolean value = TRUE
```

Procedure insert(strtoadd:string; var wheretoadd:string; startpos:integer);

يُضيف هذا الروتين السلسلة strtoadd إلى السلسلة wheretoadd ابتداء من الموقع(الدليل) startingpos و في حال أن السلسلة الناتجة غدا طولها أكبر من 255 رمزاً فإن أول 255 فإن أول 255 رمزاً سوف تحفظ و يتم تجاهل الباقي.

مثال:افترض=تخيل أن المتحول mystring له القيمة there, duckiest فإن العبارة التالية :

Insert('Hello ',mystring,1);

سوف تجعل المتحول mystring يملك القيمة Hello there, duckiest.

و البرنامج التالي يبين لنا عمل الإجراء insert:

Code

```
program test;
const emptystring='';
var val,toadd:string;
startindex:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
```

```
end;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
getstring('string (Enter to stop)? ',val);
while val<>emptystring do
begin
getstring('string to add? ',toadd);
getinteger('starting position? ',startindex);
insert(toadd,val,startindex);
writeln('result = ',val);
getstring('string (Enter to stop)? ',val);
end;
end.
```

خرج البرنامج السابق قد يكون شبيه بالتالي:

```
string (Enter to stop)?  syrian arab republic
string to add?  khaled
starting position?  1
result = khaled syrian arab republic
string (Enter to stop)?  syrian arab republic
string to add?  (khaled)
starting position?  2
result = s(khaled)yrian arab republic
string (Enter to stop)?
```

```
Function INT(val:real):real;
```

يعيد هذا التابع القسم الصحيح من العدد الحقيقي val الممرر إليه و لكنه يعيده حقيقياً فعلى سبيل المثال:إذا كانت قيمة المتحول myval=69.5645 عندئذ سوف تلحق العبارة التالية result:=INT(myreal) القيمة 69.0 بالمتحول result.

و البرنامج التالي يوضح لنا عمل التوابع frac و int و trunk :

Code

```
program test;
var val,result:real;
s:integer;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
repeat
getreal('value (> 500 to stop)? ',val);
result:=INT(val);
writeln('INT ( ',val:8:3,')   = ',result:4:1);
result:=frac(val);
writeln('FRAC ( ',val:8:3,')  = ',result:4:3);
```

```
s:=trunc(val);
writeln('TRUNC ( ',val:8:3,') = ',s:3);
until val>500.0;
readln
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value (> 500 to stop)?  22.364
INT (   22.364)   = 22.0
FRAC (   22.364)  = 0.364
TRUNC (   22.364) =  22
value (> 500 to stop)?  47.648
INT (   47.648)   = 47.0
FRAC (   47.648)  = 0.648
TRUNC (   47.648) =  47
value (> 500 to stop)?  88.364
INT (   88.364)   = 88.0
FRAC (   88.364)  = 0.364
TRUNC (   88.364) =  88
value (> 500 to stop)?  500.212
INT (  500.212)   = 500.0
FRAC (  500.212)  = 0.212
TRUNC (  500.212) = 500
```

المهندس خالد ياسين الشيخ

```
Function length(str:string):integer;
```

يعيد هذا الروتين طول السلسلة str فإذا كانت قيمة المتحول mystr هي this long عندئذ سوف تُلحق القيمة 9 بالمتحول الصحيح result:

Result:=length(mystr);

و البرنامج التالي يشرح لنا عمل التابع length:

Code

```
program test;
const emptystring='';
var val:string;
   result:integer;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
getstring('string (Enter to stop)? ',val);
while(val <> emptystring) do
begin
result:=length(val);
writeln(val,' is ',result,' characters long');
getstring('string (Enter to stop)? ',val);
end;
```

```
end.
```

خرج البرنامج شبيه بالتالي:

```
string (Enter to stop)?  Syrian Arab Republic
Syrian Arab Republic is 20 characters long
string (Enter to stop)?  khaled yassin alsheikh
khaled yassin alsheikh is 22 characters long
string (Enter to stop)?
```

```
Function PI:real;
```

يعيد هذا الروتين القيمة π و التي تساوي تقريباً 3.14156265 و هذا التابع مفيد جداً عندما نستخدم التوابع المثلثية cos و sin .

```
Function POS(strtofind, strtosearch :string):integer;
```

يبحث هذا الروتين عن السلسلة strtofind ضمن السلسلة strtosearch و في حال العثور عليها فإن هذا التابع يعيد موقع بداية أول ظهور لهذه السلسلة أما إذا لم يعثر على هذه السلسلة فإنه يعيد القيمة 0 .
مثلاً: افترض=تخيل أن قيمة المتحول pattern هي 'for' و قيمة المتحول source هي 'california' عندئذ ستلحق العبارة القيمة 5 بالمتحول الصحيح result:

Result:=pos(pattern,source);

و البرنامج يوضح عمل التابع pos:

Code

```
program test;
const emptystring='';
var val,tofind:string;
   result:integer;
   f:text;
   const m='c:\f.txt';
procedure getstring(message:string; var value:string);
begin
write(f,message,' ');
readln(value);
writeln(f,value);
end;
begin
assign(f,m);
rewrite(f);
getstring('string (Enter to stop)? ',val);
while(val <> emptystring) do
begin
getstring('string to find? ',tofind);
result:=pos(tofind,val);
writeln(f,'starting position = ',result);
getstring('string (Enter to stop)? ',val);
end;
```

```
close(f);
end.
```

```
string (Enter to stop)?  Syrian Arab Republic
string to find?  Arab
starting position = 8
string (Enter to stop)?  khaled yassin alsheikh
string to find?  sheikh
starting position = 17
string (Enter to stop)?
```

المهندس خالد ياسين الشيخ

```
Function RANDOM :real;
Function RANDOM(maxval:integer):integer; :maxval>=0;
Function RANDOM(maxval:longint):longint; :maxval>=0;
Function RANDOM(maxval:word):word;
```

يعيد هذا التابع قيمة عشوائية random value و يمكن استدعاء التابع بدون متحولات وسيطية و عندئذ يعيد هذا التابع قيمة حقيقية عشوائية بين 0 و 1 .
أما إذا استدعينا التابع السابق مع متحول وسيطي صحيح n موجب فإن التابع يولد قيمة عشوائية من 0 إلى n-1.

```
Procedure randomize;
```

يهيئ هذا الروتين مولد الأعداد العشوائية مع بذور قيم عشوائية و هذه البذور تخزن ضمن المتحول المسبق التعريف Randseed . يجب استدعاء هذه الإجراء في بداية أي برنامج يستخدم الأعداد العشوائية من خلال التابع random. أما إذا لم نستدع هذا الإجراء فإن هذا تسلسل الأعداد العشوائية المتولدة سيعتمد على القيمة المخزنة ضمن المتحول randseed . و في حال كنا نريد أن نولد لها أعداداً لها نفس التسلسل العشوائي في كل تنفيذ للبرنامج علينا تهيئة المتحول randseed بنفس القيمة عوضاً عن استدعائنا للإجراء randomize.
و البرنامجين التالي يوضحان ذلك:
البرنامج بدون استدعائنا للإجراء randomize:

Code

```
program test;
var s,i:longint;
w:word;
begin
randseed:=10;
i:=1;
while(i<=5)do
begin
s:=random(15);
write(s,',');
i:=i+1;
end;
writeln;
for i:=1 to 5 do
begin
s:=random(15);
write(s,',');
```

```
end;
writeln;
readln
end.
```

البرنامج الثاني مع استدعائنا للإجراء randomize:

Code

```
program test;
var s,i:longint;
w:word;
begin
randomize;
i:=1;
while(i<=5)do
begin
s:=random(15);
write(s,',');
i:=i+1;
end;
writeln;
for i:=1 to 5 do
begin
s:=random(15);
write(s,',');
end;
writeln;
readln
end.
```

الفرق بين البرنامجين أن البرنامج الأول سيقوم بطباعة القيم العشوائية نفسها في كل مرة ننفذ فيها البرنامج أم البرنامج الثاني سيقوم بطباعة قيم عشوائية مختلفة في كل مرة ننفذ فيها البرنامج.

```
Function sizeof(type of variable name):word;
```

يعيد هذا التابع كمية الذاكرة المخصصة لمتحول أو لنوع معطيات و هذا الناتج يمثل عدد البايتات المخصصة للمتحول. و البرنامج التالي يوضح لنا كمية الذاكرة المخصصة لكل نوع من أنواع المعطيات المعرفة ضمن البرنامج:

Code

```
program test;
type
info=record
m:integer;
s:string;
end;
var I:integer;
BY:byte;
L:longint;
sh:shortint;
W:word;
c:char;
```

```
B:boolean;
Str:string;
R:real;
begin
writeln('Integer  : ',sizeof(integer):3,' bytes; ',' I   : ':3,sizeof(I):3,' bytes');
writeln('Byte     : ',sizeof(Byte):3,' bytes; ',' By  : ':3,sizeof(By):3,' bytes');

writeln('Shortint : ',sizeof(Shortint):3,' bytes; ',' Sh  : ':3,sizeof(Sh):3,' bytes');

writeln('Longint  : ',sizeof(Longint):3,' bytes; ',' I   : ':3,sizeof(Longint):3,' bytes');


writeln('Word     : ',sizeof(Word):3,' bytes; ',' W   : ':3,sizeof(W):3,' bytes');
writeln('CHAR     : ',sizeof(CHAR):3,' bytes; ',' C   : ':3,sizeof(C):3,' bytes');

writeln('Boolean  : ',sizeof(Boolean):3,' bytes; ',' B   : ':3,sizeof(B):3,' bytes');
writeln('String   : ',sizeof(String):3,' bytes; ',' Str : ':3,sizeof(Str):3,' bytes');
writeln('Real     : ',sizeof(Real):3,' bytes; ',' R   : ':3,sizeof(R):3,' bytes');
writeln('info     : ',sizeof(info):3);
readln;
end.
```

خرج البرنامج السابق هو:

```
Integer  :   2 bytes;  I   :   2 bytes
Byte     :   1 bytes;  By  :   1 bytes
Shortint :   1 bytes;  Sh  :   1 bytes
Longint  :   4 bytes;  I   :   4 bytes
Word     :   2 bytes;  W   :   2 bytes
CHAR     :   1 bytes;  C   :   1 bytes
Boolean  :   1 bytes;  B   :   1 bytes
String   : 256 bytes;  Str : 256 bytes
Real     :   6 bytes;  R   :   6 bytes
info     : 258
```

المهندس خالد ياسين الشيخ

```
Function upcase (ch:char):char;
```

يعيد هذا الروتين الحرف الكبير للحرف الممرر إليه ضمن المتحول الوسيطي. و البرنامج التالي يوضح لنا عمل التابع upcase:

Code

```
program test;
var val:string;
index, Lgth:integer;
const emptystring='';
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
```

```
begin
getstring('string (Enter to stop)? ',val);
while(val<>emptystring)do
begin
write(val,' : ');
lgth:=length(val);
for index:=1 to lgth do
val[index]:=UPCASE(val[index]);
writeln(val);
getstring('string (Enter to stop)? ',val);
end;
end.
```

خرج البرنامج السابق هو شبيه بالتالي:

```
string (Enter to stop)?  syrian arab republic
syrian arab republic : SYRIAN ARAB REPUBLIC
string (Enter to stop)?  khaled yassin alsheikh
khaled yassin alsheikh : KHALED YASSIN ALSHEIKH
string (Enter to stop)?  arabic syria
arabic syria : ARABIC SYRIA
string (Enter to stop)?  Arab Syria
Arab Syria : ARAB SYRIA
string (Enter to stop)?
```

استدعاء الإجراءات و التوابع calling procedure and function:

عند ترجمة الملف المصدري source file (البرنامج) يقوم المترجم compiler بترجمة شفرة البرنامج code كلها من البداية و حتى النهاية و لكي تنتهي عملية الترجمة بنجاح يجب أن تنتمي عبارات البرنامج إلى الصيغ الكتابية لعبارات لغة البرمجة (لغة باسكال مثلاً) و هذا القانون ينطبق بالحتمية على الإجراءات و التوابع مونها عبارات من عبارات لغة باسكال لذلك ينبغي تعلم صيغ الروتينات و التأقلم مع صرامتها.

يمكن أن نذكر ضمن الحديث عن صرامة الصيغ الكتابية syntax هنا أن لائحة المتحولات الوسطية argument list التي تستخدم عند استدعاء الإجراء أو التابع يجب أن تطابق لائحة الوسطاء parameter list ضمن رأس الروتين المستدعى و حتى تتم عملية الترجمة بنجاح و ينجز المترجم استدعاءات الروتينات يجب أن يتعرف المترجم على رؤوس الروتينات routine head.

لنفترض مثلاً: أن لدينا إجراءان A و B و كلا الإجراءين معروف ضمن البرنامج الرئيسي main program إذا كان الإجراء A يستدعي الإجراء B عندئذ يجب على المترجم أن يعرف أو يقرأ إن صح التعبير رأس الإجراء B حتى يخمن التركيب الصحيح لاستدعاء هذا الإجراء و هذا يعني بالضرورة أن الإجراء B يجب أن يُعرف قبل الإجراء A ضمن قسم التصريح للبرنامج الرئيسي. و لكن افترض=تخيل أن الإجراء B يستدعي الإجراء A أيضاً هل يمكن ذلك ؟ حتى يتمكن الإجراء B من استدعاء الإجراء A يجب أن يكون A مُعرف قبل الإجراء B و لكن ما العمل إذا توجب علينا أن يكون A قبل B و B قبل A ؟ .

من أجل ذلك تمكننا لغة باسكال من استخدام التصريح الأمامي forward لتعريف رؤوس إجراءات أو توابع سيأتي تعريفها لاحقاً ضمن البرنامج و ذلك بأن نذكر رأس الروتين متبوعاً بفاصلة منقوطة و بعدها نضع الكلمة المحجوزة forward متبوعة بفاصلة منقوطة و هذا الترتيب يُخبر المترجم بأن لا يتوقع أن يأتي بعد رأس الإجراء الجزء المتبقي من الروتين أي قسم التصريح و جسم الروتين و كمثال على ذلك لنأخذ رأس الإجراء التالي:

Procedure Check_values (val1,val2:real); forward;

وفيما بعد و في قسم التصريح يجب أن نضع التصريح الفعلي لهذا الإجراء متضمناً رأس الإجراء (لكن بدون لائحة وسطاء) وقسم الإجراء و قسم الإجراء فمثلاً يمثل السرد التالي التصريح الفعلي عن الإجراء check_values بعد أن صرحنا عنه بالمميز المحجوز forward آنفاً. (إذاً يمكننا عدم وضع لائحة الوسطاء لأننا وضعناها في سطر forward).

Procedure check_values; (*لا حلجة إلى لائحة الوسطاء لأن المترجم قد قرأها سابقاً*)
(*هنا قسم التصريح *)
Begin

(*هنا جسم الإجراء *)
End;

الروتينات المتداخلة nested routines:

تملك الإجراءات و التوابع بنية البرنامج نفسها أي {اس و قسم تصريح و جسم فكما نستطيع ضمن البرنامج الرئيسي أن نعرف إجراءات و توابع يمكننا ضمن الروتينات أن نعرف روتينات أخرى مطلقين على الروتين الذي نعرفه ضمن روتين آخر روتين متداخل nested routine مع روتين خارجي.

يقدم البرنامج مثالاً عن الروتينات المتداخلة وقم بتتبع تنفيذ هذا البرنامج من اجل فهم عملية استدعاء الروتينات وراقب المكدس stack بالضغط على المفتاحين ctrl+f3 ضمن نقاط نقاط مختلفة من البرنامج لرؤية ما خزنته الروتينات في المكدس و لإغلاق نافذة المكدس اضغط enter فقط.

يملك الإجراء getboundedint روتينات متداخلة مهمتها مساعدته في الحصول على نتيجة المطلوبة فالإجراء adjustvals ينجز التعديل المطلوب ليصبح الحد السفلي أقل من الحد العلوي. أما التابع isbetween يفحص النتيجة المدخلة من المستخدم فيما إذا كانت ضمن الحدود المفروضة. فإذا أعاد هذا التابع القيمة true فإن حلقة repeat ستنتهي.

Code

```
program test;
var value:integer;
procedure getboundedint(message:string; minval,maxval:integer;var value:integer);
procedure adjustvals(var small,large:integer);
var temp:integer;
begin
writeln('start adjustvals');
if small>large then
begin
temp:=small;
small:=large;
large:=temp;
end;
writeln('finish adjustvals');
end;

function isbetween (val,small,large:integer):boolean;
begin
writeln('start isbetween');
isbetween:=(small<=val) and (large>=val);
writeln('finish isbetween');
end;
begin
writeln('start getboundedint');
adjustvals(minval,maxval);
repeat
write(message,' ');
readln(value);
until isbetween (value,minval,maxval);
writeln('finish getboundedint');
end;
begin
writeln('start main program');
getboundedint('value between -10 and 55:',-10,55,value);
writeln(value);
writeln('finish main program');
```

```
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
start main program
start getboundedint
start adjustvals
finish adjustvals
value between -10 and 55: 100
start isbetween
finish isbetween
value between -10 and 55: -12
start isbetween
finish isbetween
value between -10 and 55: 28
start isbetween
finish isbetween
finish getboundedint
28
finish main program
```

قضايا تصميمية برمجية program design issues:

إن إمكانية بناء روتينات متداخلة تمكننا من إنشاء إجراءات و توابع مستقلة عن بعضها و تحفظنا من الفوضى في عمل البرنامج الرئيسي فعلى سبيل المثال افترض=تخيل أننا نرغب بتصميم برمجيات SOFTWARE فعالة و نريد من برامجنا أن تكون قوية قدر الإمكان عندئذ ينبغي أن نعالج كل حالات الدخل الممكنة فإذا كان برنامجنا يتوقع قيم دخل عددية و أدخل المستخدم خطأً سلسلة رمزية فإن تنفيذ البرنامج سيتوقف و لتجنب هذه المشكلة علينا أن ننشئ روتيناً يقرأ المعلومات على أنها سلسلة رمزية ومن ثم يحول هذه السلسلة إلى قيم عددية وفق ترتيبها المناسب.
إن مثل هذا الروتين الذي يقبل قيماً تنتمي إلى عدة أنواع معطيات و يقبل أيضاً إدخالات خاطئة يمكننا إنشاؤه بواسطة روتينات متداخلة يجزئ أعمال الروتين الأصلي و كمثال على ذلك دعنا نعمل على بناء هذا الإجراء من أجل نوع المعطيات عدد صحيح integer و خوارزمية هذا الإجراء هي تحويل السلسلة الرمزية رمزا رمزا إلى قيم عددية و بناء قيمة واحدة من هذه القيم.يتوقف الإجراء عندما يصادف إدخالاً خاطئاً أو عندما يصل إل نهاية السلسلة الرمزية و هذا الإجراء سوف ينجز التالية مراراً و تكراراً :

1. يفحص كون الرقم التالي رقماً.
2. فإذا كان هذا الرمز رقماً فإنه يوجد القيمة العددية لهذا الرمز و إلا فإنه ينهي تنفيذ الحلقة.
3. إذا تمت قراءة رقم آخر يضرب بالعدد 10 و يضاف الناتج إلى القيمة الكلية .
4. يزداد العداد.

يتوجب على الإجراء أيضاً ان يفحص الحالات الخاصة منها كون الرمز الأول إحدى الإشارتين اساب minus - أو الموجب plus +
يحتوي البرنامج التالي على الإجراء makeint و الذي يمثل هذه الخوارزمية. نفذ هذا البرنامج وفق قيم صحيحة و قيم خاطئة فمثلا نفذ البرنامج من أجل القيم 5 و -100 و 62kh و khaled . و يتألف هذا الإجراء من ثلاثة روتينات متداخلة معه يجزئ الأعمال الموكلة إليه :

Code
program test; var str:string; result:integer; procedure makeint(str:string; var result:integer); const base=10;errorval=0; var index,howlong:integer; negative:boolean; function isdigit(c:char):boolean;

```
begin
isdigit:=(c>='0')and (c<='9');
end;
function toval(c:char):integer;
begin
toval:=ord(c)-ord('0');
end;
procedure handlefirstch;
begin
if(str[1]='-')then
begin
negative:=true;
inc(index);
end
else
if str[1]='+' then
inc(index);
end;
begin
result:=0;
negative:=false;
howlong:=length(str);
if(howlong>0)then
begin
index:=1;
handlefirstch;
while(index<=howlong)do
begin
if isdigit(str[index])then
begin
result:=base*result+toval(str[index]);
inc(index);
end
else
index:=howlong+1;
end;

if negative then
result:=result*-1;
end
else
result:=errorval;
end;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
repeat
```

```
getstring('value? (>500 to stop) ',str);
makeint(str,result);
writeln(str,' : ',result);
until result>500;
readln;
end.
```

خرج البرنامج السابق قد يكون شبيه بالتالي:

```
value? (>500 to stop)  khaled
khaled : 0
value? (>500 to stop)  62kh
62kh : 62
value? (>500 to stop)  66k88
66k88 : 66
value? (>500 to stop)  -59
-59 : -59
value? (>500 to stop)  200
200 : 200
value? (>500 to stop)  5001
5001 : 5001
```


قراءة الملفات المصدرية أثناء الترجمة reading source files during compilation:

يملك الروتين makeint طريقة قوية في الحصول على دخل مُؤلف من أعداد صحيحة من المستخدم و تختلف هذه الطريقة تماما عن الطريقة المستخدمة في الإجراء getinteger إذ لن يتوقف تنفيذ البرنامج إذا أدخل المستخدم خطأً حرفاً عوضاً عن عدد صحيح . إذاً قد نحتاج إلى كتابة هذا الإجراء ضمن البرامج التي تحتاج إلى دخل عبارة عن عدد صحيح. إذاً قد نحتاج إلى كتابة هذا الإجراء ضمن الكثير من البرامج و لكن هذه العملية ستكون طويلة و مملة في حين أننا نستطيع و ضع الإجراء makeint ضمن ملف مستقل و إذا أردنا استخدام هذا الروتين ضمن برامجنا يمكننا إخبار المترجم بأن يقرأ محتويات هذا الملف و يضعه في المكان المناسب من البرنامج الذي نعمل فيه حسب العمل و المكان المناسب و ذلك باستخدام توجيهات المترجم compiler directive و هي عبارة عن تعليمات مُوجهة للمترجم حتى ينجز بعض الأعمال الجزئية المعنية.
مثال: افترض= تخيل أنه لدينا ملف اسمه test2.pas يحتوي على الإجراءين makeint و getinteger عندئذ ستمثل الأسطر التالية محتويات هذا الملف و لاحظ التعديلات التي أجريت على الإجراء getinteger حتى يتمكن من الحصول على قيمة عددية باستخدام استدعاء الإجراء makint كما يلي:

Code for file "test2.pas"

```
procedure makeint(str:string; var result:integer);
const base=10;errorval=0;
var index,howlong:integer;
negative:boolean;
function isdigit(c:char):boolean;
begin
isdigit:=(c>='0')and (c<='9');
end;
function toval(c:char):integer;
begin
toval:=ord(c)-ord('0');
end;
procedure handlefirstch;
begin
```

```
if(str[1]='-')then
begin
negative:=true;
inc(index);
end
else
if str[1]='+' then
inc(index);
end;
begin
result:=0;
negative:=false;
howlong:=length(str);
if(howlong>0)then
begin
index:=1;
handlefirstch;
while(index<=howlong)do
begin
if isdigit(str[index])then
begin
result:=base*result+toval(str[index]);
inc(index);
end
else
index:=howlong+1;
end;

if negative then
result:=result*-1;
end
else
result:=errorval;
end;
procedure getinteger(message:string; var value:integer);
var tempstr:string;
begin
write(message,' ');
readln(tempstr);
makeint(tempstr,value);
end;
```

بعد أن أنشأنا الملف test2.pas يمكننا كتابة برنامج يستخدم محتويات الملف test2.pas كالتالي مثلاً:

Code
program test; (*$I test2.pas *) var intval:integer;

```
procedure name;
begin
getinteger('value? ',intval);
writeln(sqr(intval));
end;
begin
name;
getinteger('value? ',intval);
writeln(sqrt(intval):6:2);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
value?  12kh14
144
value?  625
 25.00
```

المهندس خالد ياسين الشيخ

يُوعز توجيه المترجم السابق (* test2.pas $I*) للمترجم بقراءة محتويات الملف المذكور test2.pas في تلك النقطة من الملف و هذا يكافئ تماماً كتابة محتويات هذا الملف ضمن البرنامج و في موضع توجيه المترجم حيث يُضيف المترجم محتويات هذا الملف و يترجم البرنامج بشكل كامل و الصيغة الكتابية لتوجيه المترجم هذا يمكن أن تكتب:

```
(* اسم الملف  $I*)
        أو:
{ اسم الملف $I}
```

ملاحظة note: يجب ان يأتي توجيه المترجم مباشرة بعد رمز التعليق أي ينبغي ان لا يتواجد فراغ بين رمز التعليق و إشارة الدولار $ و إلا فإن المترجم لن ينفذ التوجيه .

العودية أو التراجعية أو الاستدعاء الذاتي Recursion:

لقد تعرفنا سابقاً في هذا الكتلب كيف يُنقل التحكم بين البرنامج و الروتينات التي يستدعيها إذ يتوقف تنفيذ الروتين مؤقتاً حتى ينفذ روتين آخر تم استدعه من قبل الروتين الحالي و من ثم يعود التحكم لتنفيذ باقي تعليمات الروتين. فعلى سبيل المثال إذا استدعى البرنامج الرئيسي إجراء اسمه first فإن البرنامج سوف يتوقف تنفيذه مؤقتاً و يبدأ تنفيذ الإجراء first و إذا كان هذا الإجراء يستدعي إجراء آخر اسمه second فإن تنفيذ أجراء first سيتوقف مؤقتاً أيضاً حيث أصبح لدينا الآن روتينان قد توقف تنفيذهما بشكل مؤقت و ينفذ الإجراء second و إذا استدعى الإجراء second إجراء ثالثاً اسمه third فإن الإجراء second بدوره سيتوقف و ينفذ الإجراء third فإذا لم يستدع الإجراء third أي إجراء آخر فسيكمل تنفيذه كلياً و عند الانتهاء من تنفيذه يعود التحكم إلى الإجراء second و بعد الانتهاء من تنفيذ الإجراء second يعود التحكم إلى الإجراء الذي استدعاه و هو first و الإجراء first بعد الانتهاء من تنفيذ تعليماته يُعيد التحكم إلى البرنامج الرئيسي .

ما يهمنا من هذه التذكرة هنا أن لا يغيب عن ذاكرتنا أن الإجراءات و التوابع يعاد تحفيزها عكسياً reactive backward إذ إن آخر توقف مؤقت طبق على الروتين سيقتضي بأن يكون أول إعادة تحفيز للروتين و هذه الخاصية القوية و المفيدة تستخدم في عمل الروتينات العودية recursive routine و هي الروتينات التي تستدعي نفسها أي يتوقف تنفيذ الروتين المستدعى بشكل مؤقت و يتم تحفيز نفس الروتين (المستدعي) و للروتينات العودية وظيفة معينة فإذا أنجزت هذه الوظيفة في خطوة واحدة فإن الروتين يعيد التحكم إلى الروتين المستدعي و إلا فإن الروتين يُمرر هذه الوظيفة بعد تبسيطها إلى نفس الروتين لإكمال تنفيذ هذه المهمة و بعبارة أبسط: كل روتين عودي يستدعي نفسه ممرراً الوظيفة الموكلة إليه بعد تبسيطها حتى يتم حلها بعد عدد من الخطوات.

و **الخلاصة**: العودية تعتمد على مفهوم المكدس stack.

Code

```
program test;
var level,countval:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
procedure count(val:integer);
begin
level:=level+1;
writeln(' ':level*2,'satarting count, level = ',level:2,'; val = ',val);
if(val>1)then
count(val-1);
writeln(' ':level*2,'leaving count, level = ',level:2,'; val = ',val);
level:=level-1;
end;
begin
repeat
level:=0;
getinteger('count to what to value? (0 to stop)',countval);
if countval>0 then
count(countval);
until countval=0;
readln;
end.
```

خرج البرنامج السابق من أجل countval=6 هو كما يلي:

```
count to what to value? (0 to stop) 6
  satarting count, level =  1; val = 6
    satarting count, level =  2; val = 5
      satarting count, level =  3; val = 4
        satarting count, level =  4; val = 3
          satarting count, level =  5; val = 2
            satarting count, level =  6; val = 1
            leaving count, level =  6; val = 1
          leaving count, level =  5; val = 2
        leaving count, level =  4; val = 3
      leaving count, level =  3; val = 4
    leaving count, level =  2; val = 5
  leaving count, level =  1; val = 6
count to what to value? (0 to stop) 0
```

المهندس خالد ياسين الشيخ

يُظهر البرنامج السابق الخرج بطريقة تساعدنا على معرفة ما يحدث ضمن المستويات العودية المختلفة و الجدول التالي يوضح هذه العمليات و الكلمة main تعني البرنامج الرئيسي:

الروتين المستدعي	السطر المطبوع	الروتين الذي توقف تنفيذه
Main		
count(6)	starting count, level=1; val=6	main
count(5)	starting count, level=2; val=5	main
		count(6)
count(4)	starting count, level=3; val=4	main
		count(6)
		count(5)
count(3)	starting count, level=4; val=3	main
		count(6)
		count(5)
		count(4)
count(2)	starting count, level=5; val=2	main
		count(6)
		count(5)
		count(4)
		count(3)
count(1)	starting count, level=6; val=1	main
		count(6)
		count(5)
		count(4)
		count(3)
		count(2)

نلاحظ من خلال الجدول السابق أن تتابع الاستدعاء الذاتي للإجراء count يتم مع قيم أصغر للمتحول و يتوقف هذا التتابع من الاستدعاءات عندما يصل إلى الاستدعاء count(1) لأن الإجراء count عند القيمة 1 يُنفذ بدون أي استدعاء ذاتي آخر أي يُظهر الإجراء count عندها العبلوة الابتدائية و النهائية مباشرة لأنه لا يقوم بأي استدعاء ذاتي كما يلي:

starting count, level = 6; val=1
leaving count, level = 6; val=1

و بعد انتهاء الاستدعاء الذاتي يبد الاستداء الإجراء بشكل عكسي:

الروتين المتوقف	السطر المطبوع (النهائية)	الروتين المستدعي
Main		
Count(6)		
Count(5)		
Count(4)		
Count(3)		
Count(2)	Leaving count, level=6; val=1	count(1) (stop recursion)
Main		
Count(6)		
Count(5)		
Count(4)		
Count(3)	Leaving count, level=5, val=2	count(2)
Main		
Count(6)		
Count(5)		
Count(4)	Leaving count, level=4, val=3	count(3)
Main		
Count(6)		
Count(5)	Leaving count, level=3, val=4	count(4)
Main		
Count(6)	Leaving count, level=2, val=5	count(5)
main	Leaving count, level=1, val=6	count(6)
		main

انسياب الاستدعاء العودي أو التراجعي the flow of a recursive call:

لقد لاحظنا في البرنامج السابق العلاقة بين استدعاء الروتين و إكماله حيث أن آخر روتين يستدعى هو أول روتين ينهى و أول روتين يستدعى يكون آخر روتين ينهى و نلاحظ من خلال الإجراء count أن الوسيط مرر بقيمته passed by value أي أن كل إجراء يحتفظ بقيمة val الخاصة به و هذه الخاصية تمكننا من تنفيذ إجراءات مختلفة و بقيم مختلفة.

خواص الروتين العودي features of a recursive routine:

إذا جردنا الإجراء السابق count ليصبح كالتالي:

```
Procedure count(val:integer);
Begin
  If(val>1)then
     Count(val-1);
  Writeln(val);
End;
Begin {main}
  Count(3);
End;
```

خرج البرنامج أعلاه هو:

1
2

3

فالروتين العودي يجب أن يتمتع بالصفات التالية:

1. أن يتضمن الروتين شرط لتحديد فيما إذا كان الروتين يستطيع إنهاء عمهله ببساطة أم يحتاج إلى استدعاء نفسه مرة أخرى. و في مثالنا الشرط هو if val>1
2. أن يتم ضمن الروتين و بعد فحص الشرط تغيير لقيمة الوسطاء بحيث يتم يضمن لنا الروتين إنهاء العودية (حسب عمل كل روتين).
3. يمكن أن نضع ضمن الروتين العودي عبارات الطباعة أو القراءة أو شروط تحكم ....الخ.

ملاحظة: العودية مكلفة زمنياً ومادياً للحاسب.

من حلات عدم توقف العودية (الاستدعاء الذاتي) و يؤدي ذلك إلى حدوف الفيض over flow كما في المثالين التالين:

```
Procedure count(val:integer);
Begin
    Count(val-1);
    If(val>1)then
      Writeln(val);
End;
Begin{main}
Count(3);
End;

Procedure count(val:integer);
Begin
  If(val>1)then
    Count(val+1);
  Writeln(val);
End;
Begin {main}
Count(3);
End;
```

مثال آخر عن العودية Another Example:

يولد المثال التالي سلسلة فيبوناشي Fibonacci numbers مستخدماً في ذلك استدعائين عوديين ضمن نفس الروتين.
أطلق علة هذه السلسلة اسم مكتشفها ليوناردو بيزا Leonardo pisa و الذي يعرف باسم فيبوناشي Fibonacci و الذي استطاع من خلالها حل مشكلة توليد الأرانب حيث كانت المسألة المطروحة امامه كم عدد الأزواج التي تنتج في كل عما وفق المعطيات التالية:

1. نبدأ بزوج من الأرانب.
2. كل شهر يلد زوج الأرانب هذا زوجاً آخر.
3. زوج الأرانب المولود يحبح خصباً (قابل للإنجاب) بعد شهر من ولادته.

لنبدأ بزوج من الأرانب و ننتظر شهراً واحدا حتى تصبح هذ الأرانب قابلة للإنجاب أي أن أول حدين من السلسلة هما 1 و 1 أي fib(1)=1 و fib(2)=1 و ينتج الحد الثالث من هذه السلسلة من جمع الحدين السابقين له أي سنحصل في الشهر الثالث على زوجين من الأرانب هما زوج الأرانب الأصلي وزوج الأرانب المولود حديثاً أي أن:

Fib(3)=Fib(2)+Fib(1)=1+1=2

وبالتالي:

Fib(4)=Fib(3)+Fib(2)=2+1=3

أما الشهر الخامس فسوف نحصل على خمسة أزواج من الأرانب ثلاثة منها قديمة و اثنان مولودان حديثاً.أحد الأزواج المولودة آتٍ من الزوج الأصلي القديم و الزوج الثاني آتٍ من الزوج السابق أي أن:

Fib(5)=Fib(4)+Fib(3)=3+2=5

يمكن تمثيل عملية توليد الأرانب من زوج واحد وفق الجدول الآتي على افتراض أنه في الشهر الأول كان لدينا زوج من الأرانب غير مخصب و في الشهر الثاني أصبح هذا الزوج مخصب (قابل للإنجاب) وزوج الأرانب الخصب ينتج بعد شهر زوج أرانب غير مخصب لذلك رمزنا لزوج الأرانب المخصب بالرمز + و غير المخصب بالرمز –
أي ينتج الزوج + بعد شهر زوج – أي أن + يعطينا بعد شهر - + حيث يرمز + لزوج الأرانب نفسه و – للزوج المولود حديثاً

حيث الزوج غير الخصب يُخصب بعد شهر أي يعطينا – بعد شهر + و الجدول الممثل لهذه العلاقة هو كالتالي:

الشهر	التوالد و الإخصاب عند الأرانب في كل شهر	عدد أزواج الأرانب
1	-	1
2	+	1
3	+-	2
4	+-+	3
5	+-++-	5
6	+-++-+-+	8
7	+-++-+-++-++-	13

يمكن بسهولة إتمام هذا الجدول حى يشمل العدد المطلوب من الأشهر و بنفس الطريقة.و لا يقتصر استخدام سلسلة فيبوناشي على عملية توليد الأرانب فقط بل تستخدم في حساب نمو النباتات حيث تخضع كثير من النباتات لهذه السلسلة الشهيرة.
يمكننا الآن دراسة هذه السلسلة رياضياً كما يلي:

1. الحد الأول و الثاني يساويان الواحد فرضاً أي أن :

Fib(1)=1 and fib(2)=1

2. كل حد من حدود هذه السلسلة يساوي ناتج جمع الحدين السابقين له في هذه السلسلة أي أن:

Fib(n)=Fib(n-1)+Fib(n-2)

و الجدول التالي يحتوي على أول سبعة عشر حداً من حدود هذه السلسلة.

number	value	Source
Fib(1)	1	Definition
Fib(2)	1	Definition
Fib(3)	2	Fib(1)+Fib(2)
Fib(4)	3	Fib(2)+Fib(3)
Fib(5)	5	Fib(3)+Fib(4)
Fib(6)	8	Fib(4)+Fib(5)
Fib(7)	13	Fib(5)+Fib(6)
Fib(8)	21	Fib(6)+Fib(7)
Fib(9)	34	Fib(7)+Fib(8)
Fib(10)	55	Fib(8)+Fib(9)
Fib(11)	89	Fib(9)+Fib(10)
Fib(12)	144	Fib(10)+Fib(11)
Fib(13)	233	Fib(11)+Fib(12)
Fib(14)	377	Fib(12)+Fib(13)
Fib(15)	610	Fib(13)+Fib(14)
Fib(16)	987	Fib(14)+Fib(15)

و البرنامج التالي يحوي الإجراء Fib الذي يعمل على توليد أعداد سلسلة فيبوناشي Fibonacci:

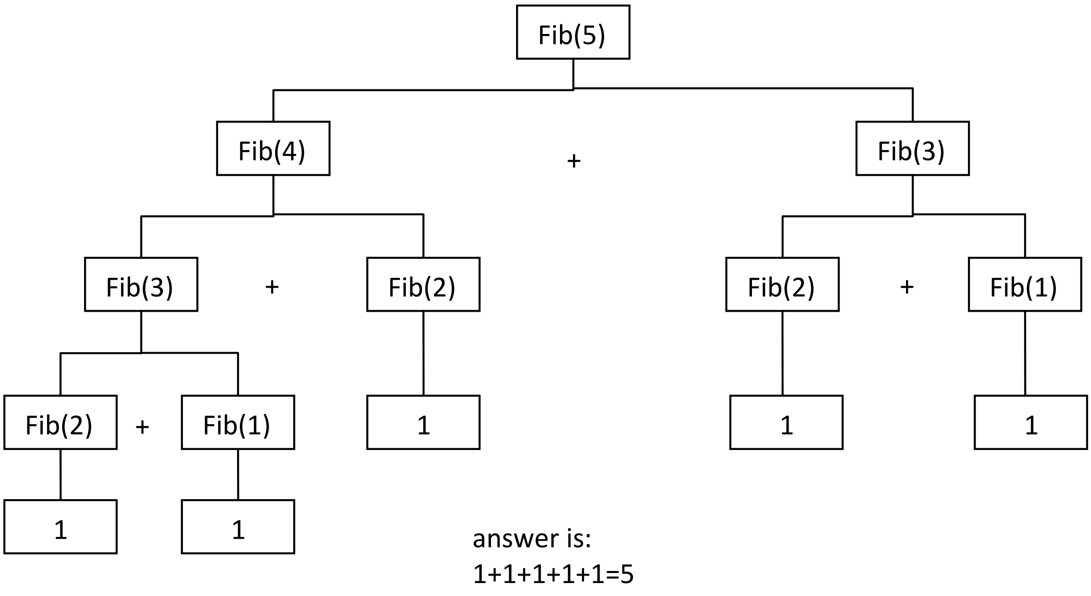
Code
```
program test;
var val:integer;
function Fib(val:integer):longint;
begin
if(val<=0)then
Fib:=0
else
if(val=1)or(val=2)then
Fib:=1
else
Fib:=Fib(val-1)+Fib(val-2);
end;
var i:integer;
begin
for i:=1 to 20 do
writeln('Fib(',i:2,' )=',Fib(i):4);
readln;
end.
```

خرج البرنامج السابق هو:

```
Fib( 1 )=   1
Fib( 2 )=   1
Fib( 3 )=   2
Fib( 4 )=   3
Fib( 5 )=   5
Fib( 6 )=   8
Fib( 7 )=  13
Fib( 8 )=  21
Fib( 9 )=  34
Fib(10 )=  55
Fib(11 )=  89
Fib(12 )= 144
Fib(13 )= 233
Fib(14 )= 377
Fib(15 )= 610
Fib(16 )= 987
Fib(17 )=1597
Fib(18 )=2584
Fib(19 )=4181
Fib(20 )=6765
```

المهندس خالد ياسين الشيخ

يحسب التابع Fib حدود سلسلة فيبوناشي باستخدام الاستدعاءات العودية فإذا كانت قيمة المتحول الممرر إليه val تساوي قيمة سالبة او تساوي الصفر فإن التابع سوف ينتهي مباشرة وبدون أي استدعاء عودي ويعيد القيمة 0 مباشرة.
و كذلك الأمر إذا كانت قيمة val الممرر تساوي 1 أو 2 سوف ينتهي التابع مباشرة ويعيد القيمة 1 و إلا فإن هذا التابع سوف يقسم المشكلة إلى استدعائين ذاتيين أحدهما يحسب القيمة Fib(val-1) و الثاني يحسب القيمة Fib(n-2) و هذان الاستدعاءان بدورهما يُقسمان المشكلة و يبسطانها إلى حين الوصول إلى القيمة 1 أو 2 و الشكل التالي يبين لنا مخطط المنطقي=التخيلي=الافتراضي ... للاستدعاءات من أجل الحد الخامس:



نلاحظ قوة العودية و سرعتها في الوصول إلى النتائج المطلوب و كمثال عن تعقيد (كلفة) العودية :إذا عدلنا البرنامج السابق بحيث من حدود سلسلة فيبوناشي و يحسب عدد الاستدعاءات العودية في كل حد كما يلي:

Code

```
program test;
var val:integer;
calls:longint;
function Fib(val:integer):longint;
begin
if(val<=0)then
Fib:=0
else
if(val=1)or(val=2)then
Fib:=1
else
begin
calls:=calls+1*2;
Fib:=Fib(val-1)+Fib(val-2);
end;
end;

var i:integer;
begin
calls:=0;
for i:=1 to 30 do
begin
writeln('Fib(',i:2,' )=',Fib(i):6,calls:10,' calls to Fib');
calls:=0;
end;
readln;
```

```
end.
```

خرج البرنامج السابق هو:

```
Fib( 1 )=      1         0 calls to Fib
Fib( 2 )=      1         0 calls to Fib
Fib( 3 )=      2         2 calls to Fib
Fib( 4 )=      3         4 calls to Fib
Fib( 5 )=      5         8 calls to Fib
Fib( 6 )=      8        14 calls to Fib
Fib( 7 )=     13        24 calls to Fib
Fib( 8 )=     21        40 calls to Fib
Fib( 9 )=     34        66 calls to Fib
Fib(10 )=     55       108 calls to Fib
Fib(11 )=     89       176 calls to Fib
Fib(12 )=    144       286 calls to Fib
Fib(13 )=    233       464 calls to Fib
Fib(14 )=    377       752 calls to Fib
Fib(15 )=    610      1218 calls to Fib
Fib(16 )=    987      1972 calls to Fib
Fib(17 )=   1597      3192 calls to Fib
Fib(18 )=   2584      5166 calls to Fib
Fib(19 )=   4181      8360 calls to Fib
Fib(20 )=   6765     13528 calls to Fib
Fib(21 )=  10946     21890 calls to Fib
Fib(22 )=  17711     35420 calls to Fib
Fib(23 )=  28657     57312 calls to Fib
Fib(24 )=  46368     92734 calls to Fib
Fib(25 )=  75025    150048 calls to Fib
Fib(26 )=121393     242784 calls to Fib
Fib(27 )=196418     392834 calls to Fib
Fib(28 )=317811     635620 calls to Fib
Fib(29 )=514229    1028456 calls to Fib
Fib(30 )=832040    1664078 calls to Fib
```


نلاحظ من خلال الخرج السابق أنه من أجل تغير طفيف في حدود السلسلة أي تغيير بسيط في قيمة المتحول val سيرافقه تغيير كبير في عدد الاستدعاءات العودية(التراجعية).

الحلول التكرارية للمشاكل العودية iterative solutions To Recursive problem:

قد تكون الحلول غير العودية nonrecursive أو التكرارية لبعض السائل أو المشاكل أسرع و أبسط لأن كل المسائل أو المشاكل يمكن أن تحل بالطرق التكرارية (من الناحية النظرية) فعلى سيل المثال يمكننا حل مشكلة فيبوناشي بالطريقة التكرارية كما يلي:

Code

```
program test;
var val:integer;
calls:longint;
function Fib(val:integer):longint;
var running,smaller,larger,count:longint;
begin
if(val<=0)then
begin
Fib:=0;
exit;
end;
if(val=1)or(val=2)then
fib:=val
```

```
else
begin
smaller:=1;
larger:=1;
count:=3;
repeat
calls:=calls+1;
running:=smaller+larger;
smaller:=larger;
larger:=running;
inc(count);
until count>val;
fib:=running;
end;
end;
var i:integer;
begin
for i:=1 to 30 do
begin
writeln('Fib(',i:2,')=',Fib(i):6,calls:10,' calls to Fib');
calls:=0;
end;
readln;
end.
```

خرج البرنامج السابق هو:

```
Fib( 1)=     1         0 sum to Fib
Fib( 2)=     2         0 sum to Fib
Fib( 3)=     2         1 sum to Fib
Fib( 4)=     3         2 sum to Fib
Fib( 5)=     5         3 sum to Fib
Fib( 6)=     8         4 sum to Fib
Fib( 7)=    13         5 sum to Fib
Fib( 8)=    21         6 sum to Fib
Fib( 9)=    34         7 sum to Fib
Fib(10)=    55         8 sum to Fib
Fib(11)=    89         9 sum to Fib
Fib(12)=   144        10 sum to Fib
Fib(13)=   233        11 sum to Fib
Fib(14)=   377        12 sum to Fib
Fib(15)=   610        13 sum to Fib
Fib(16)=   987        14 sum to Fib
Fib(17)=  1597        15 sum to Fib
Fib(18)=  2584        16 sum to Fib
Fib(19)=  4181        17 sum to Fib
Fib(20)=  6765        18 sum to Fib
Fib(21)= 10946        19 sum to Fib
Fib(22)= 17711        20 sum to Fib
Fib(23)= 28657        21 sum to Fib
Fib(24)= 46368        22 sum to Fib
Fib(25)= 75025        23 sum to Fib
Fib(26)=121393        24 sum to Fib
Fib(27)=196418        25 sum to Fib
Fib(28)=317811        26 sum to Fib
Fib(29)=514229        27 sum to Fib
Fib(30)=832040        28 sum to Fib
```

المهندس خالد ياسين الشيخ

نلاحظ من خلال مقارنة خرج البرنامجين السابقين أن الطريقة التكرارية ستكون أسرع مقارنة مع الطريقة العودية (طبعا كلما كان حد السلسلة أكبر).
أن الطريقة التكرارية أطول كتتابع تعليمات لكنها أسرع تنفيذاً في حين أن الطريقة العودية على العكس من ذلك.

و يكفي أن نقول على سبيل المثال لا الحصر أن الطرية العودية في حساب استدعاء Fib(30) تحتاج إلى 1,664078 (أي ما يقارب مليوني استدعاء) في حين أن الطرية التكرارية تحتاج إلى بضع عشرات من عمليات الجمع.

اكتب تابع عودي rsum يقوم بحساب العلاقة التالية:

F(0)=0
F(n)=F(n-1)+n: n>0

مع توضيح عملية الاستدعاءات من أجل n=5 ؟.

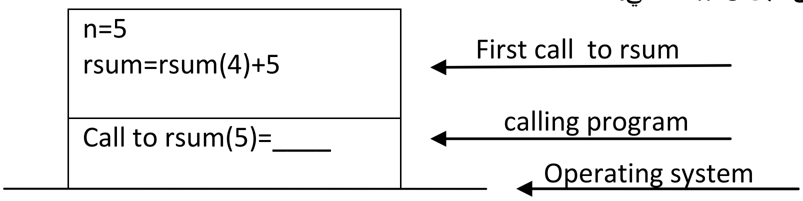
Code
program test; function rsum(n:integer):integer; begin if(n<1)then rsum:=0 else rsum:=rsum(n-1)+n; end; var n,i:integer; begin write('enter the value number integer>=1...'); readln(n); write('total 1 to ',n:3,'=',rsum(n):4); readln end.

خرج البرنامج السابق من أجل n=100 هو:

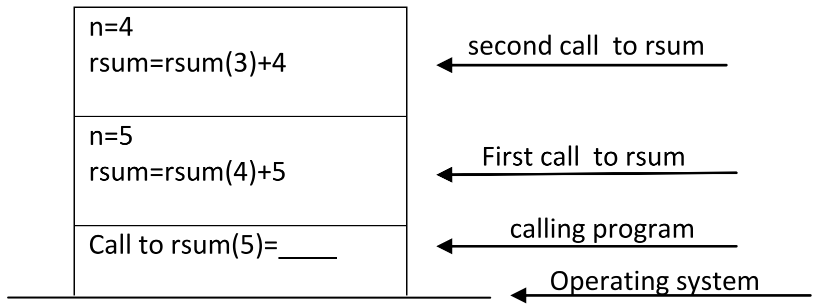
```
enter the value number integer>=1...100
total 1 to 100=5050
```

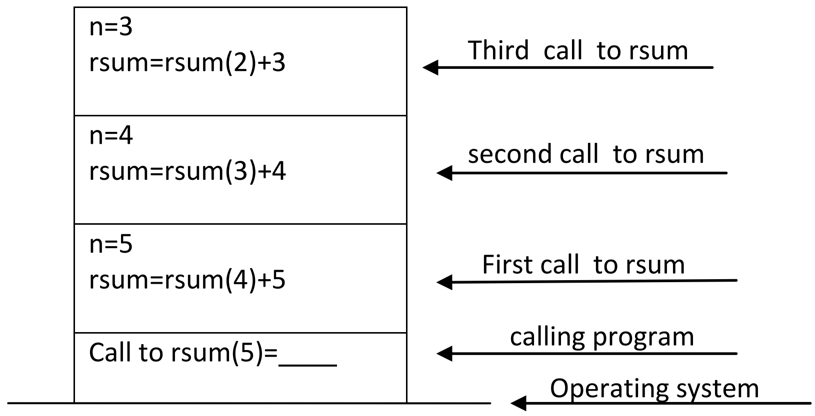
المهندس خالد ياسين الشيخ

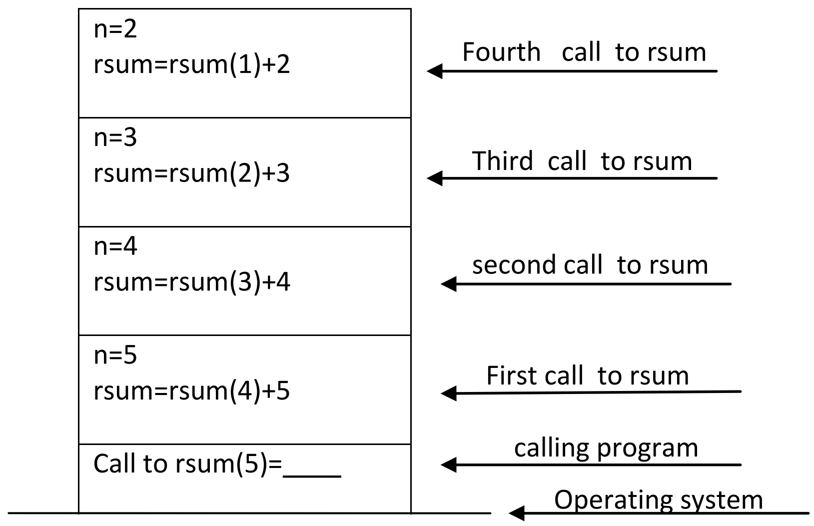
توضح عملية الاستدعاءات من أجل n=5 كالتالي:

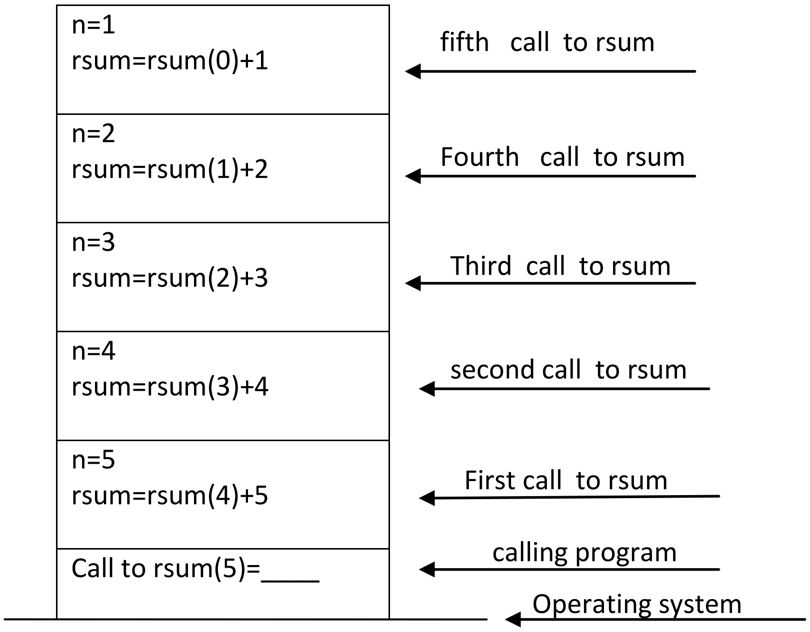


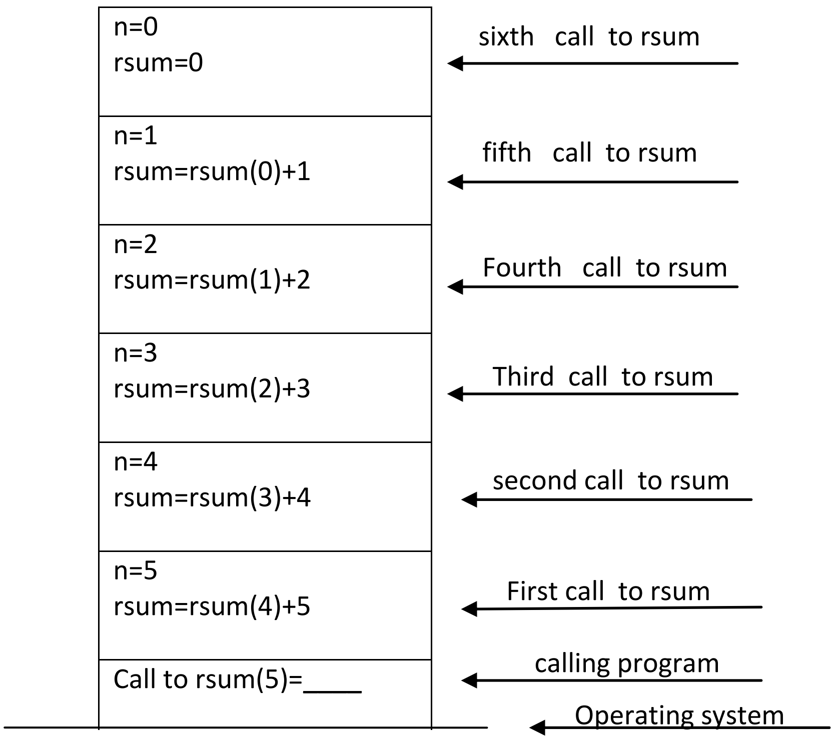
ملاحظة هامة جداً: كل استدعاء عودي ينسخ متحولاته الخاصة بالذاكرة RAM .


n=4
rsum=rsum(3)+4
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
second call to rsum
First call to rsum
calling program
Operating system

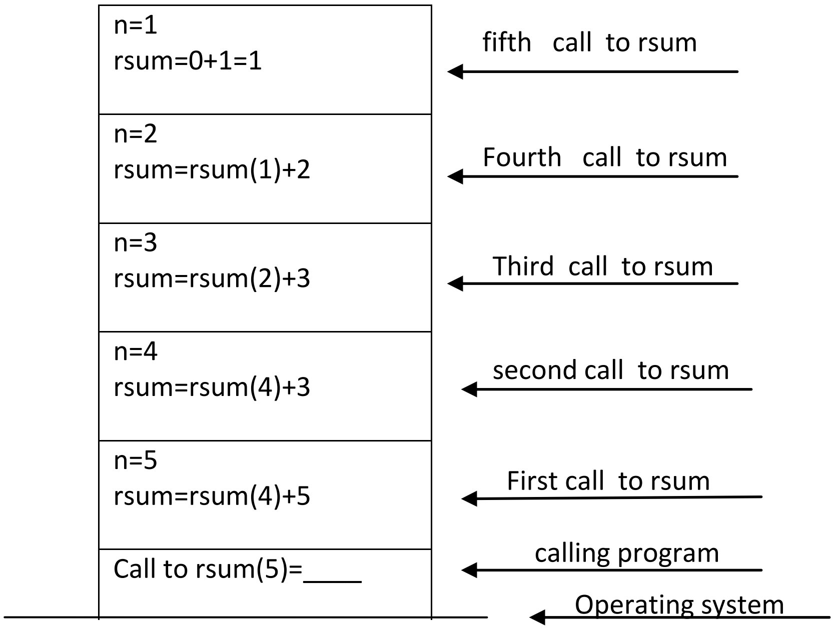

n=3
rsum=rsum(2)+3
n=4
rsum=rsum(3)+4
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
Third call to rsum
second call to rsum
First call to rsum
calling program
Operating system

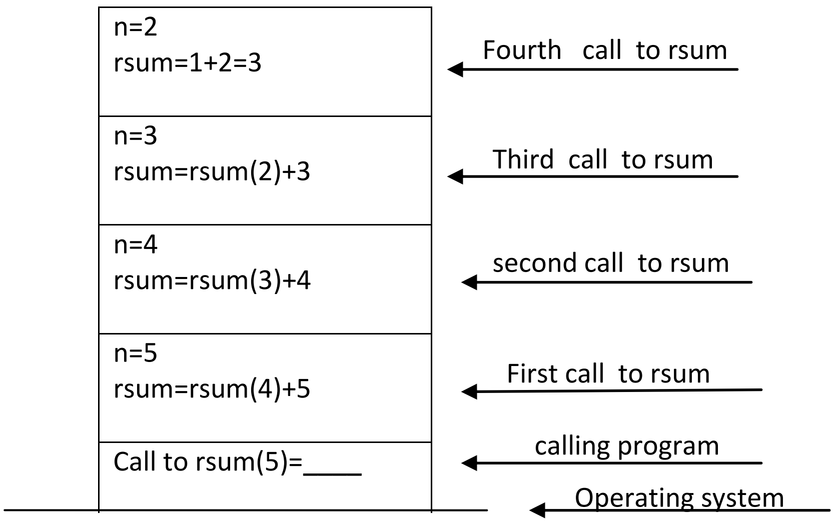

n=2
rsum=rsum(1)+2
n=3
rsum=rsum(2)+3
n=4
rsum=rsum(3)+4
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
Fourth call to rsum
Third call to rsum
second call to rsum
First call to rsum
calling program
Operating system

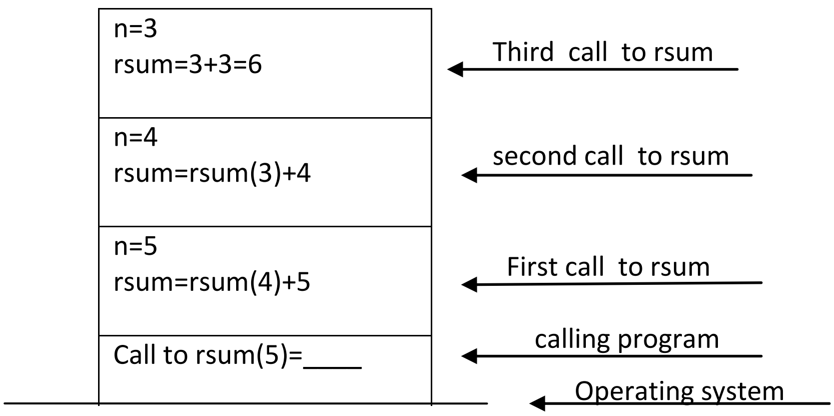


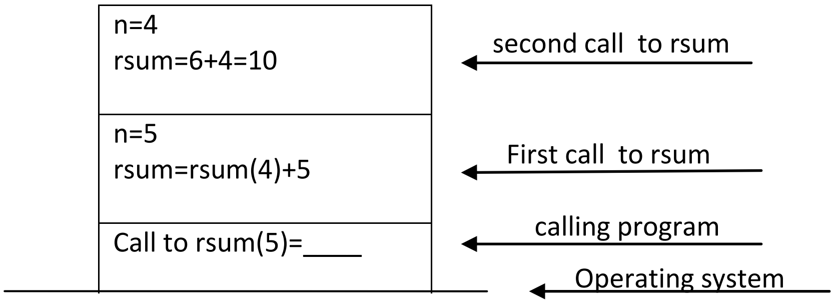


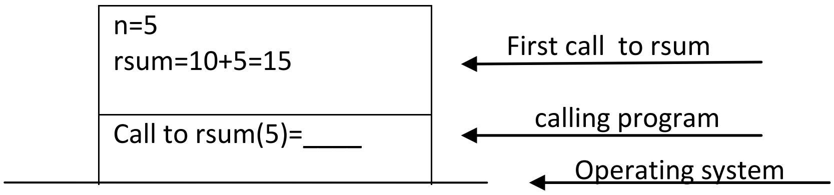
و الآن نعود بشكل معاكس (تراجعي) (العودية تعتمد على مفهوم المكدس) :

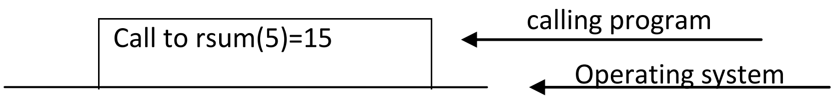

n=1
rsum=0+1=1
n=2
rsum=rsum(1)+2
n=3
rsum=rsum(2)+3
n=4
rsum=rsum(4)+3
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
fifth call to rsum
Fourth call to rsum
Third call to rsum
second call to rsum
First call to rsum
calling program
Operating system


n=2
rsum=1+2=3
n=3
rsum=rsum(2)+3
n=4
rsum=rsum(3)+4
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
Fourth call to rsum
Third call to rsum
second call to rsum
First call to rsum
calling program
Operating system


n=3
rsum=3+3=6
n=4
rsum=rsum(3)+4
n=5
rsum=rsum(4)+5
Call to rsum(5)=____
Third call to rsum
second call to rsum
First call to rsum
calling program
Operating system







اكتب تابع عودي Factoriel لحساب العلاقة التالية:

n!=1 if n=1
otherwise n!=1*2*3*4...........*n(n-2)*(n-1)*n

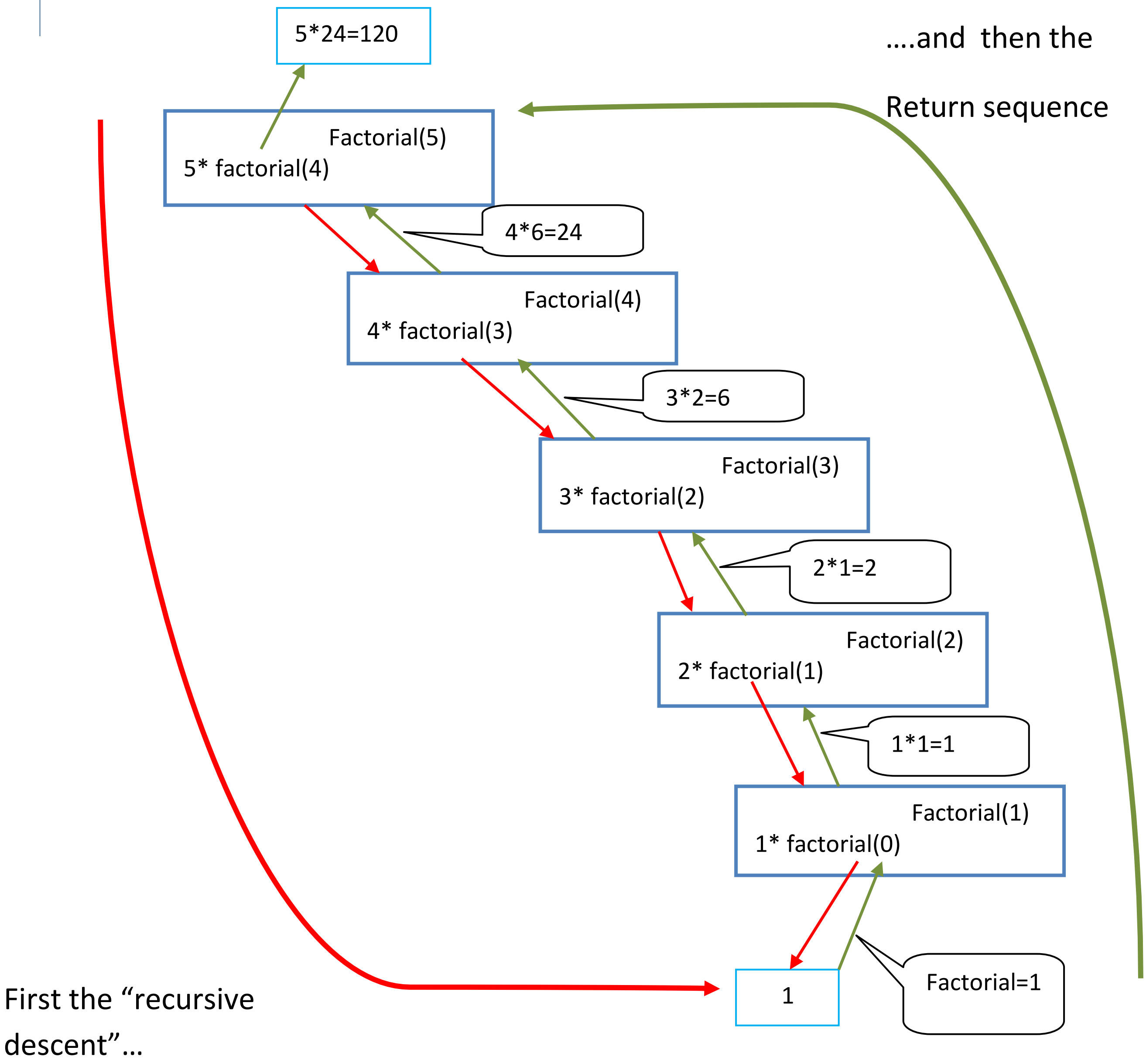
مع توضيح عملية الاستدعاءات العودية (recursive execution trace) من اجل Factoriel(5):

Code

```
program test;
function factorial(n:integer):longint;
begin
if(n>0)then
factorial:=n*factorial(n-1)
else
factorial:=1;
end;
var i:integer;
begin
for i:=0 to 8 do
writeln(i,'!=':2,factorial(i));
readln
end.
```

خرج البرنامج السابق هو:

```
0!=1
1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5040
8!=40320
```

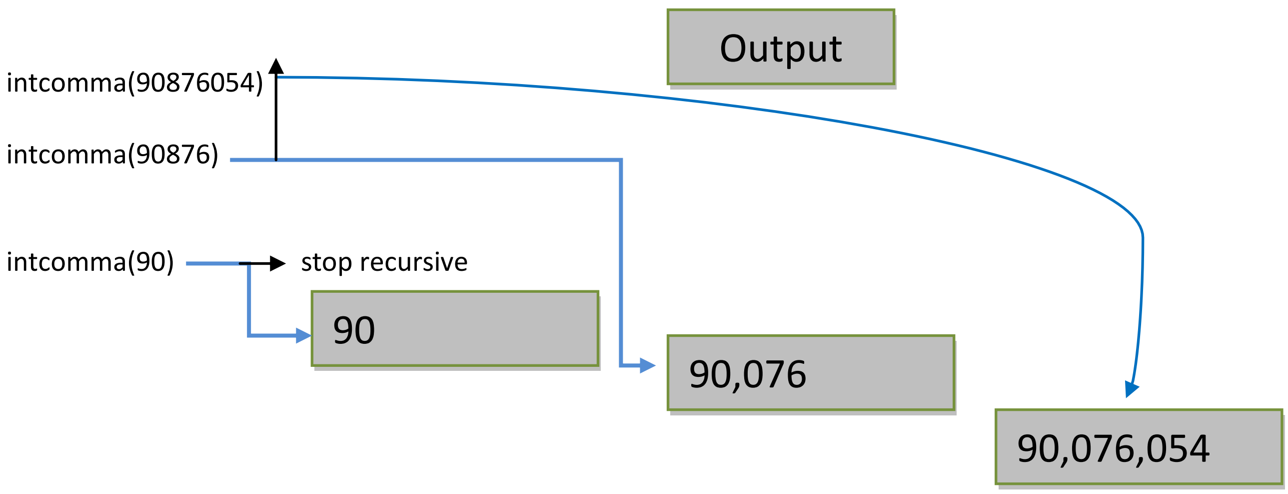
<u>وضح خرج البرنامج التالي مع توضيح عملية الاستدعاءات :</u>

Code

```
program test;
procedure intcomma(n:longint);
begin
if(n<0)then
begin
write('-');
n:=-n;
end
else
if(n<1000)then
write(n:5)
else
begin
intcomma(n div 1000);
write(',');
n:=n mod 1000;
write(n div 1000);
n:=n mod 100;
write(n div 10,n mod 10);
end;
end;
begin
intcomma(90876054);
writeln;
readln;
end.
```

خرج البرنامج السابق هو:

```
90,076,054
```



اكتب تابع عودي وسمه sum_prime لإيجاد مجموع العوامل الأولية لعدد صحيح موجب تماماً .
اكتب إجرائية print لطباعة مجاميع العوامل الأولية لعدد صحيح موجب تماماً.
مثال: من أجل العدد 15 يتم طباعة:

5+3=8

من أجل العدد 150 يتم طباعة:

2+3+5+5=15

Code

```
program test;
function sum_prime(n,i:integer):integer;
begin
if(n=1)then
sum_prime:=0
else
if(n mod i=0)then
sum_prime:=sum_prime(n div i,i)+i
else
sum_prime:=sum_prime(n,i+1);
end;
procedure print(n:integer);
var i:integer;
begin
for i:=2 to n do
while(n mod i=0)do
begin
if(i<>n)then
write(i,'+')
else
write(i,'=');
n:=n div i;
end;
end;
var n:integer;
begin
write('enter the value integer>0....');
readln(n);
print(n);
writeln(sum_prime(n,2));
readln;
end.
```

خرج البرنامج السابق من أجل n=250 هو:

```
enter the value integer>0....250
2+5+5+5=17
```

بنى التحكم يمكن تقسيمها بشكل عام إلى قسمين:

1. بنى اختيار selection construct و فيها يتم تنفيذ اعمال جزئية معينة إذا تحقق الشرط الموجود في هذه البنية و يمكم أن تكون بسيطة أو مركبة.
2. بنى تكرار iteration construct و يجري من خلالها تكرار أعمال معينة عدداص محدداً من المرات أو حتى يتحقق شرط معين. و يمكن أن تكون بسيطة أو مركبة.

التكرار البسيط – عبارة For : simple repepetition -the for statement :

تستخدم حلقة for في لغة باسكال عندما نعلم مسبقاً عدد المرات التي نريد فيها تكرار العمل.

مثلاً ليكن لدينا البرنامج التالي:

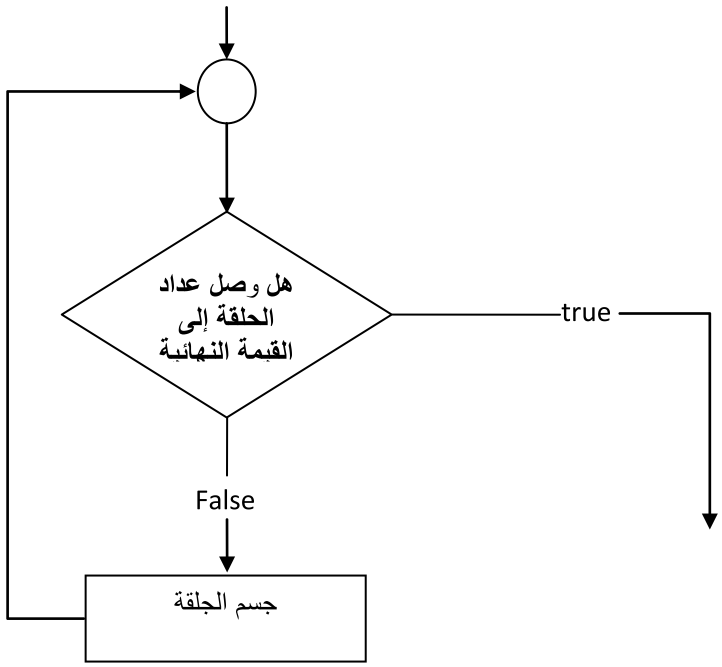
Code

```
program test;
const maxvals=10;
maxitems=5;
var index,result:integer;
procedure  dispvariants(value:integer);
const maxvariants=15;
var count:integer;
begin
write(value:5);
for count:=11 to maxvariants do
write(value+random(count*100):5);
writeln;
end;
procedure putheadings;
var count:integer;
begin
write(' ':12,'val ');
for count:=1 to maxitems*5 do
write('-');
writeln;
end;
begin {main program}
randomize;
putheadings;
for index:=1 to maxvals do
begin
write(‘0…’,index*100-1:4,':');
result:=random(index*100);
dispvariants(result);
end;
readln
end.
```

خرج البرنامج السابق هو شبيه بالتالي:

```
              val -------------------------
0...   99:     8  422 1147  769  407 1356
0...  199:    98  117  551  954 1440  976
0...  299:   253  777  476 1390  811 1126
0...  399:   126  501 1126  472  503 1086
0...  499:   186  329  365  889 1470 1239
0...  599:   316  679 1224  945 1545 1648
0...  699:   452 1253 1193 1569  903 1167
0...  799:   405 1038 1409 1274  947 1419
0...  899:   284  689  865 1029  285  880
0...  999:   770  995 1579 1978  936 1863
```

يولد البرنامج السابق عشرة أعداد صحيحة عشوئية بحيث يقع العدد الأول بين 0 و 99 والعدد الثاني بين 0 و 199 و هكذا حتى آخر قيمة عشوائية التي سوف تكون بين 0 و 999 وهذه الأعداد الناتجة سوف تكتب على أسطر مسقلة تحت كلمة val و لاحظ أن حلقة for تحوي عباة مركبة compound statement (لأنها تكرر مجموعة من التعليمات ضمن جسم الحلقة في كل مرة يتم فيها تنفيذ الحلقة) و استدعاء الإجراء dispvariants في السطر الأخير من حلقة for يكتب خمس قيم عشوائية صحيحة في كل سطر و كل واحدة من هذه القيم ناتجة عن مجموع أول قيمة في السطر مع قيمة عشوائية أخرى فاول قيمة عشوائية تتراوح بين 0 و 1099 و القيمة الثانية تتراوح بين 0 و 1199 و هكذا حتى نصل إلى آخر قيمة و هي تتراوح بين 0 و1499 .
و كما لاحظنا فإن الإجراء dispvariants يحوي حلقة for بسيطة حيث تحتوي على عبارة واحدة تتكرر أما الإجراء writeln فيستدعى بعد أن تتكرر العبارة البسيطة عدد من المرات.
و يحتوي الإجراء putheading على حلقة for بسيطة ترسم سطراً من الرمز – عددا من المرات.
يمكننا تمثيل حلقة for بالمخطط الانسيابي أو المنطقي=التخيلي=الافتراضي التالي:



والصيغة الكتابية syntax لحلقة for ضمن تربو باسكال كالتالي:

متحول الحلقة =القيمة الابتدائية لمتحول الحلقة to القيمة النهائية لمتحول الحلقة do

يُسمى متحول الحلقة loop variable في حلقة for بمتحول التحكم control variabl و هو الذي يحدد عدد التكرارات و هو يمثل أيضاً عداداً للحلقة و هذا المتحول يمكن ان ينتمي إلى أي نوع من أنواع المعطيات المرتبة و لكن نوع المعطيات الأكثر استخداماً بشكل عام لمتحول الحلقة هو نوع المعطيات الصحيح.
أما القيمة الابتدائية و القيمة النهائية لمتحول الحلقة فيمكن أن تكون متحولاً أو قيمة أو تعبيراً و لكن الشيء المهم أن تكون من نفس النوع المعطيات الذي يتنتمي إليه متحول الحلقة و كمثال على ذلك :

```
Var index:integer;
For index:=1 to 10 do
```

هنا متحول الحلقة هو index و يأخذ قيم من 1 و حتى 10 و هي من نوع معطيات صحيح.
ومما يجب ملاحظته أننا لسنا بحاجة لتغيير قيمة متحول حلقة for لأن البرنامج و بشكل تلقائي يغير قيمة متحول الحلقة بعد كل تكرار لجسم الحلقة بمقدار 1 و في الحقيقة إن التغيير اليدوي لمتحول الحلقة سوف يسبب أخطاء في تسلسل تنفيذ البرنامج و البرنامجين التاليين يبين لنا هذا الخطأ:

Code

```
program test;
var i:integer;
begin
for i:=1 to 5 do
write(i,',');
writeln;
readln;
end.
```

خرج البرنامج السابق هو:

1,2,3,4,5,

Code

```
program test;
var i:integer;
begin
for i:=1 to 5 do
begin
write(i,',');
inc(i,2);
end;
readln;
end.
```

يسبب البرنامج السابق الدخول بحلقة لا نهائية بسبب تغيير قيمة المتحول الحلقة i ضمن جسم الحلقة.
وستكون قيمة i هي:

i=1,3,4,6,7,9,10,12,.........................

و بالتالي قيمة متحول عداد الحلقة i لن يصل إلى القيمة النهائية له و هي هنا 5 مما يسبب في الخول في الحلقة لا منتهية.
ملاحظة: ضمن حلقة for يتم تنفيذ محتوى جسم الحلقة ومن ثم اختبار وصول العداد إلى القيمة النهائية ثم تتم بعدها زيادة مقدار المتحول بمقدار 1 في حال عدم وصول العداد إلى القيمة النهائية.

ضمن حلقة for يتم تنفيذ جسم الحلقة و من ثم اختبار وصول عداد الحلقة إلى القيمة النهائية في لغة التربو باسكال.
و البرنامج التالي يوضح ذلك:

Code
program test; const maxsize=9; m='c:\f.txt'; var outer:char; f:text; i:integer; begin i:=200; for i:=1 to 10 do begin writeln('end',i:2,',':2); end; writeln(' ':6,'start main'); writeln('end',i:2,',':2); readln; end.

خرج البرنامج أعلاه هو:

```
end 1 ,
end 2 ,
end 3 ,
end 4 ,
end 5 ,
end 6 ,
end 7 ,
end 8 ,
end 9 ,
end10 ,
      start main
end10 ,
```

إذا كانت القيمة الابتدائية لمتحول الحلقة أكبر من القيمة النهائية لهذا المتحول فإن الحلقة لن تنفذ وأما إذا كانت القيمة الابتدائية لمتحول الحلقة تساوي القيمة النهائية لهذا المتحول فإن هذه الحلقة سوف تنفذ مرة واحدة فقط.
يجب الانتباه عند كتابة حلقة for إلى عدم كتابة الفاصلة المنقوطة ; بعد المميز المحجوز do لأن ذلك سوف يجعل الحلقة for تنفذ العبارة الفارغة null statement أي أنه لن ينفذ شيْ.
والبرنامج السابق يوضح ذلك :

Code
program test; var i:integer; begin for i:=1 to 5 do; write(i,','); writeln; readln; end.

خرج البرنامج السابق هو:

5,

الحلقات العكسية BackWard loops :

لنفترض=لنتخيل أننا نريد كتابة الأحرف الهجائية عكسي كما في السطر التالي:

t s r q p o m l k j

يمكننا فعل ذلك من خلال حلقة for العكسية حيث يتم فيها عد متحول الحلقة عكسياً و البرنامج التالي يشرح لنا هذا النوع من حلقة for . (يجب إعادة تنفيذ البرنامج عدة مرات حتى يولد قيماً عشوائية مناسبة لحلقته العكسية).

Code

```
program test;
var chindex,startch,endch:char;
procedure randomch(var thech:char);
const offset=ord('a');
var val:integer;
begin
val:=random(26)+offset;
thech:=chr(val);
end;
begin
randomize;
randomch(startch);
randomch(endch);
writeln('start = ',startch:3,' ; end = ',endch:3);
for chindex:=startch downto endch do
write(chindex:3);
writeln;
readln;
end.
```

خرج البرنامج السابق قد يكون شبيه بالتالي:

```
start =   k ; end =   a
  k  j  i  h  g  f  e  d  c  b  a
```

يولد البرنامج السابق رمزين عشوائيين يستخدم الرمز الأول كقيمة ابتدائية لمتحول الحلقة و الرمز الثاني يستخدم كقيمة نهائية لمتحول الحلقة و يُظهر البرنامج الرموز الموجودة بين هذين الرمزين ابتداء من الرمز الأول و حتى الرمز الأخير و في حال كون الرمز الأول يأتي قبل الرمز الأخير في جدول ASCII أي أن ترتيبه أقل فإن الحلقة لن تنفذ.

و يحتوي البرنامج السابق على افجراء randomch الذي يولد رمزاً عشوائياً و هذا الرمز يجب أن يكون حرفاً حيث المتحول val قد يحوي قيمة صحيحة بين 97 و 122 ثم تحول القيمة المولدة ضمن المتحول val إلى مايقابله من حرف(رمز) ضمن جدول آسكي في المتحول الحرفي thchar .

و في جسم البرنامج الرئيسي main program نجد حلقة for العكسية تنقص متحول الحلقة بمقدار رمز بعد كل تنفيذ لجسم الحلقة.

الفرق التركيبي أو للغوي بين حلقتي for الأمامية و العكسية هو المميز المحجوز downto الذي يستخدم عوضاً عن to و يجب أن تكون القيمة التي تأتي بعد downto أصغر من القيمة الابتدائية لمتحول الحلقة على عكس الحلقة الأمامية التي يجب أن تكون القيمة التي تأتي بعد to أكبر من القيمة الابتدائية لمتحول الحلقة و في حال كون القيمة النهائية لمتحول الحلقة العكسية أكبر من القيمة الابتدائية فإن جسم الحلقة لن يُنَفذ أبداً أما إذا تساوت قيمتا متحول الحلقة البدائية و النهائية فإن جسم الحلقة سينفذ مرة واحدة فقط.

حلقات forالمتداخلة nested for loops:

يمكن أن نضع أي نوع من العبارات ضمن حلقة for بما في ذلك عبارة for أيضاً فعلى سبيل المثال :افترض أننا نريد إظهار الخرج التالي:

```
a.1  a.2  a.3  a.4  a.5  a.6  a.7  a.8  a.9
b.1  b.2  b.3  b.4  b.5  b.6  b.7  b.8  b.9
c.1  c.2  c.3  c.4  c.5  c.6  c.7  c.8  c.9
d.1  d.2  d.3  d.4  d.5  d.6  d.7  d.8  d.9
e.1  e.2  e.3  e.4  e.5  e.6  e.7  e.8  e.9
f.1  f.2  f.3  f.4  f.5  f.6  f.7  f.8  f.9
g.1  g.2  g.3  g.4  g.5  g.6  g.7  g.8  g.9
h.1  h.2  h.3  h.4  h.5  h.6  h.7  h.8  h.9
i.1  i.2  i.3  i.4  i.5  i.6  i.7  i.8  i.9
j.1  j.2  j.3  j.4  j.5  j.6  j.7  j.8  j.9
k.1  k.2  k.3  k.4  k.5  k.6  k.7  k.8  k.9
```

حتى نتمكن من الحصول على الخرج السابق يجب علينا:

1. أن نكتب الحرف عدة مرات في السطر الواحد.
2. و في كل تكرار للحرف يجب أن يُرفق بنقطة و رقم متغير على طول السطر.
3. بعد عدد معين من هذا الترافق بين الحرف و الرقم يجب الانتقال إلى سطر جديد و بالتالي إلى حرف جديد.

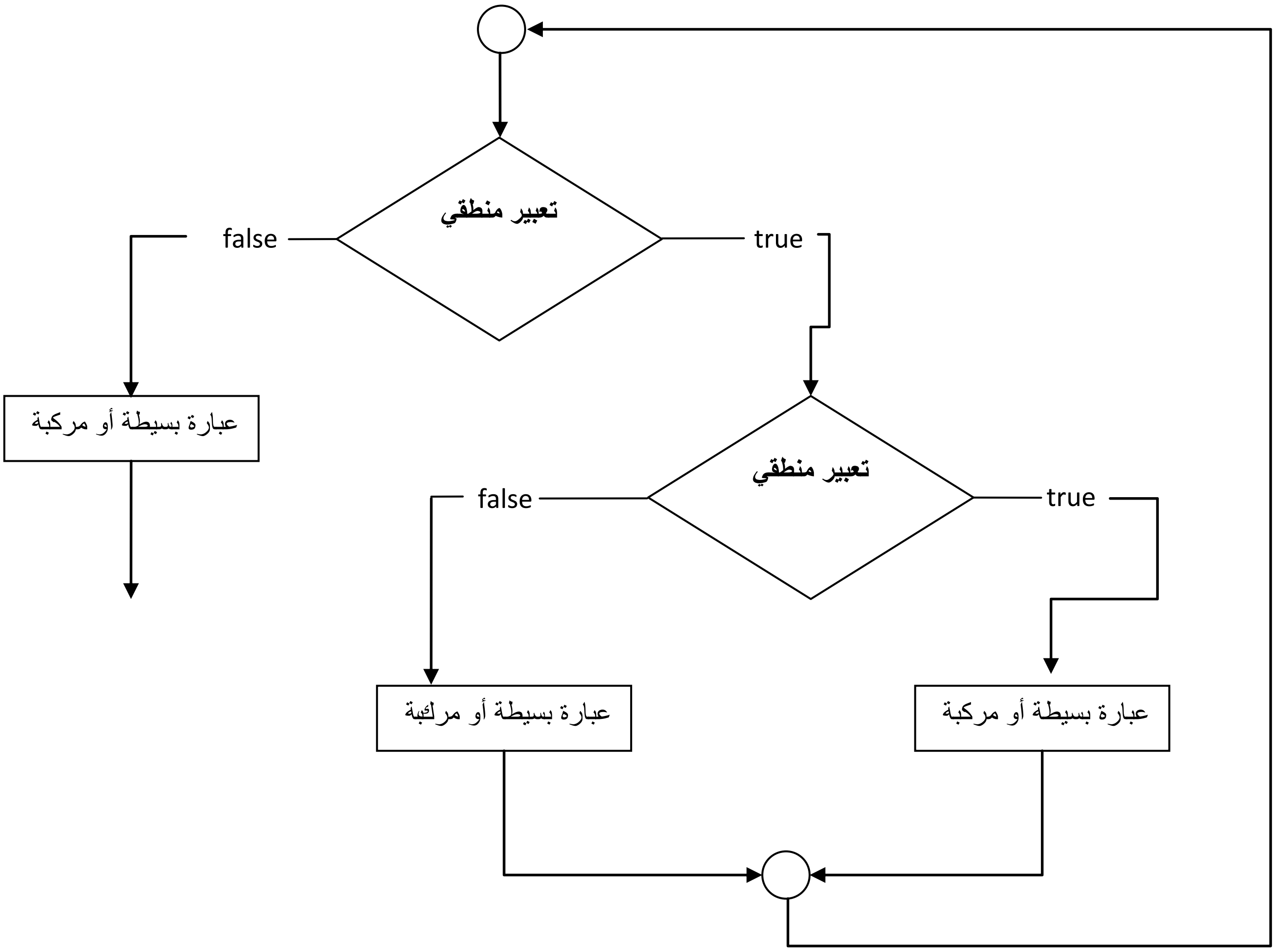
و البرنامج التالي يعطينا الخرج السابق (نستخدم حلقتي for داخلية و خارجية):

Code

```
program test;
const maxsize=9;
var outer:char;
inner:integer;
begin
for outer:='a' to 'k' do
begin
for inner:=1 to maxsize do
write(outer:3,'.',inner);
writeln;
end;
readln;
end.
```

تدعى حلقة for الداخلية التي تتحكم بها حلقة for الخارجية حلقة متداخلة nested loop و هذا يعني أن الحلقة الداخلية تنفذ بشكل كامل في كل مرة ينتهي فيها تنفيذ جسم الحلقة الخارجية.

يمكن تمثيل حلقات for المتداخلة بالمخطط الانسيابي أو المنحطقي=التخيلي=الافتراضي التالي:



و حتى يُفهم = نفهم بشكل جيد سلوك الحلقات المتداخلة علينا أن القيام ببغييرات مختلفة في هذه الحلقات و مراقبة الخرج فعلى سبيل المثال نفذ البرنامج البرنامج التالي:

Code
program test; const maxsize=3; var i,j:integer; begin for i:=1 to maxsize do begin writeln('ingress = ',i); for j:=maxsize downto 1 do writeln(i,'+',j,'=',i+j); end; writeln(' ':5,'main start'); writeln('i=',i); writeln('j=',j); readln; end.

خرج البرنامج السابق هو:

```
ingress = 1
1+3=4
1+2=3
1+1=2
ingress = 2
2+3=5
2+2=4
2+1=3
ingress = 3
3+3=6
3+2=5
3+1=4
      main start
i=3
j=1
```

المهندس خالد ياسين الشيخ

بنى الاختيار – عبارة if selection constructs – the IF statement:

قد نحتاج في برامجنا تنفيذ عمل معين في حال تحقق شرط ما أو الاستمرار في سير عمل البرنامج في حال عدم تحقق هذا الشرط فمثلاً إذا كان لدينا برنامج يبحث ضمن مجموعة من الأعداد عن قيمة معينة هل هي موجودة أم لا فإن هذا البرنامج سيقرأ كل عدد من هذه الأعداد و يقارنها مع القيمة المفروضة فإذا كان هذا العدد يساوي القيمة المطلوبة فإن البرنامج سيُعلمنا بأنه عثر على القيمة المطلوبة و إلا فإنه و ببساطة سيكرر العملية نفسها مع العدد التالي أي سيأخذ العدد التالي ضمن المجموعة و يقارنه مع القيمة المختارة و هكذا دواليك

تمنحنا عبارة if في لغة باسكال الوسيلة اتوجيه تنفيذ البرنامج اعتماداً على قرارات و شروط كما هو موضح في البرنامج التالي إذ يولد البرنامج عدداً صحيحاً عشوائياً فإذا كان هذا العدد يقبل القسمة على 10 فإن البرنامج يطبع نقطة dot على الشاشة و بعد طباعة 50 نقطة ينتقل إلى سطر جديد.

Code
program test; const maxtrials=5000; linewidth=50; divisor=10; var result,index,nrhits,nrtrials:integer; t:real; begin randomize; nrhits:=0; nrtrials:=0; for index:=1 to maxtrials do begin result:=random(maxint); if result mod 10=0 then begin write('.'); inc(nrhits,1); if(nrhits mod linewidth=0)then writeln(nrtrials:10); end; end; writeln; writeln(maxtrials,' trials,',

```
nrhits,' divsible by 10    ',nrhits/maxtrials*100:6:2,'%');
readln;
end.
```

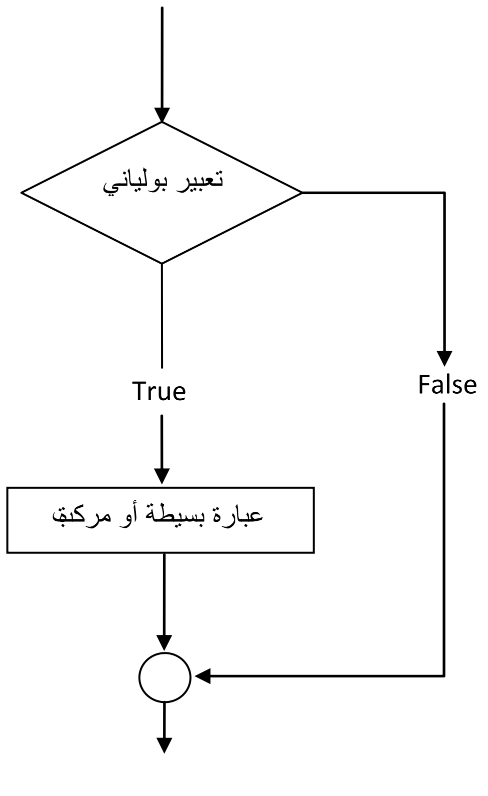
خرج البرنامج السابق شبيه بالتالي:

```
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
..................................................          0
................
5000 trials,517 divsible by 10     10.34%
```

كما رأينا في البرنامج السابق فإن الصيغة الكتابية لعبارة if كالتالي:

If < تعبير بولياني> then <عبارة مركبة أو بسيطة>

التعبير في عبارة if يجب أن يكون بوليانياً فإذا كان قيمته true فستنفذ العبارة أو العبارات التالية للمميز المحجوز then أما إذا كانت قيمته false فإن البرنامج سينتقل إلى العبارة التالية . فمثلاً في البرنامج السابق تفحص عبارة if العدد العشوائي فيما إذا كان يقبل القسمة على 10 فإذا تحقق هذا الشرط فإن العبارة المركبة الملحقة بعبارة if سوف تنفذ و إلا فإن البرنامج سينتقل إلى التعليمة التالية من تعليمات حلقة for و يمكننا تمثيل عبارة if بالمخطط الاتسيابي او المنطقي=التخيلي=الافتراضي التالي:



لاحظ أن المثال السابق يحتوي على عبارة if متداخلة و هذا أمر مسموح به تماماً كما كان في حلقات for المتداخلة و في الحقيقة يجب ملاحظة ان عبارة if الأولى متداخلة هي الآخرى مع حلقة for و بشكل عام فإن أي نوع من أنواع بنى التحكم

يمكنها لتدخل يما بينها و هذه الخاصة البسيطة تدعى بالتغليف أو الصندقة encapsulation و هي تمكن من بناء برامج متنوعة و قوية و سوف نستخدم هذا الفهوم و بشكل أعم وأوسع في البرمجة بالكائنات OOP.

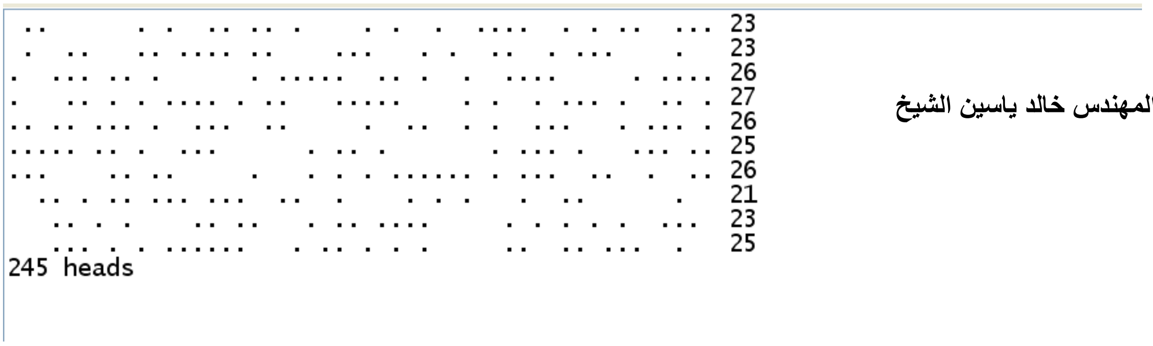
الفصل بين خيارين: عبارة if-else deciding between alternative :

قد نحتاج في بعض الأحيان بين خيارين أي إذا لا تحقق شرط ما فقم بعمل و إلا فعليك بآخر و لنتحقق عملية هذه تمكننا لغة باسكال من إضافة المميز else بعد عبارة if فتصبح هذه العبارة أكثر فائدة من عبارة if البسيطة و البرنامج التالي يشرح لنا كيفية استخدام البنيۃ if-else:

Code

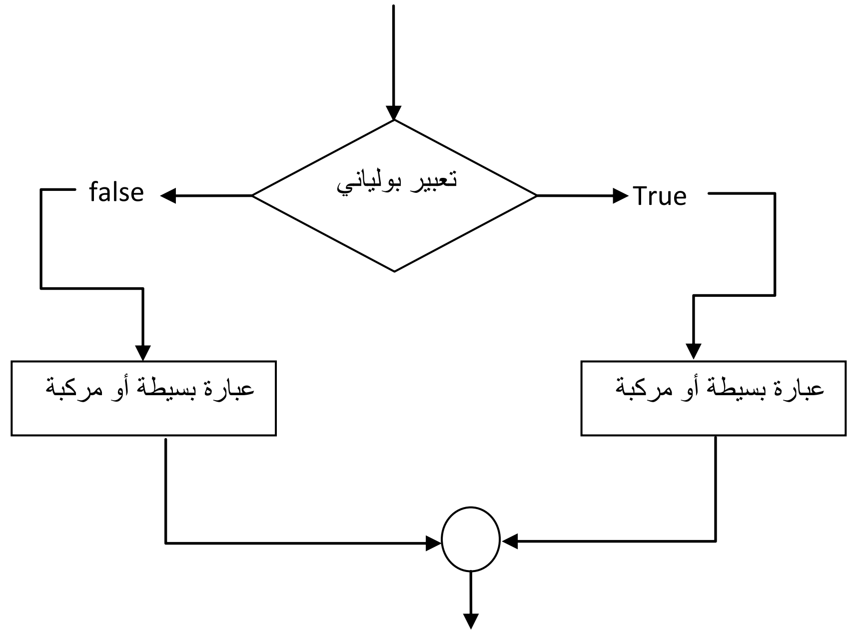
```
program test;
const maxtrials=500;
var result,count, headcount, rowcount:integer;
begin
randomize;
headcount:=0;
rowcount:=0;
for count:=1 to maxtrials do
begin
result:=random(maxint);
if result mod 2 =0 then
begin
write('.');
rowcount:=rowcount+1;
end
else
write(' ');
if count mod 50=0 then
 begin
writeln(rowcount:3);
headcount:=rowcount+headcount;
rowcount:=0;
 end;
end;
writeln(headcount,' heads');
readln;
end.
```

يُولد البرنامج السابق أعداد عشوائية صحيحة فإذا كانت هذه الأعداد زوجية سيكتب نقطة و الاسيطبع فراغ على الشاشة و بعد 50 محاولة سيكتب عدد النفاط التي كتبت في هذه السطر و من ثم ينتقل إلى سطر جديد و بعد تنفيذ البرنامج سوف نلاحظ خرجاً يشبه هذا:



المهندس خالد ياسين الشيخ

يحدد المميز المحجوز else بعده عملاً اختيارياً ينجز إذا كانت قيمة التعبير البولياني false و العبارة الآتية بعد else يمكن أن تكون بسيطة أو مركبة و الشكل المنطقي التالي يُرينا عبارة if-else:



يجب الانتباه إلى عدم وضع فاصلة منقوطة بين end التي تنهي جزء if و بين else (يجب عدم وضع فاصلة منقوطة قبل else) إذ أن هذا الاستثناء يخص بعض المواضع بلغة باسكال و التي يمنع فيها وضع فاصلة منقوطة.
حيث أن المترجم يعتبر بنية if-else عبارة واحدة و عندما نضع فاصلة منقوطة قبل else فستفصلها عن عبارة if جاعلة العبارة else عبارة مستقلة و هذا غير صحيح لأن else جزء من عبارة if و ليست عبارة مستقلة.

العمل مع البدائل:عبارة if-else الموسعة working through alternative :an extended if-else statements:
قد نحتاج في بعض الأحيان إلى اتخاذ قرارات كثيرة و متتالية و كل قرار يعتمد على إجابة القرار السابق له .و الشكل العام لمثل هذه السلسلة من القرارات هو:

```
If A then <عبارة>
        Else if B then <عبارة>
                Else if … then <عبارة>
                        Else <عبارة>
```

ومن خلال ذلك نستطيع بناء قرارات كما نريد فعلى سبيل المثال :افترض= تخيل أنه لدينا المسألة البسيطة التالية المِمثلة على شكل خوارزمية algorithm كما يلي:

1. ابدأ مع عدد صحيح موجب تماماً كيفي و ليكن A.
2. إذا كان A=1 توقف.
3. و إلا إذا كان عددا ًزوجياً استبدل A بناتج قسمته على 2 ومن ثم عد إلى الخطوة 2.
4. و إلا إذا كان عدداً فردياً A بناتج العبارة 3A+1 و عد ثانية إلى 2.

و البرنامج التالي هو تطبيق لهذه الخوارزمية (لاحظ طريقة بناء عباراتif-else الموسعة):

Code

```
program test;
var val,count,even,odd1:longint;
procedure getlongint(message:string; var value:longint);
begin
write(message,' ');
```

```
readln(value);
end;
procedure checkforfullscreen(nrlines:longint);
const fullpage=20;
begin
if(nrlines mod fullpage=0)then
begin
write('press enter to go on. ');
readln;
end;
end;
begin
writeln('Ctrl-C ends the program anytimes.');
count:=0;
even:=0;
odd1:=0;
getlongint('value?',val);
repeat
inc(count);
if(val=1)then
inc(odd1)
else
if (val mod 2=0)then
val:=val div 2
else
val:=3*val+1;
if(val<>1)then
if(val mod 2=0)then
inc(even)
else
inc(odd1);
writeln('val = ',val:8,'; odd= ',odd1:8,'; even = ',even:8);
checkforfullscreen(count);
until val=1;
writeln('After ',count:8,' iterations, result = 1.');
writeln(even:8,' even values; ',odd1:8,' odd values');
readln;
end.
```

خرج البرنامج السابق من أجل val=13 هو:

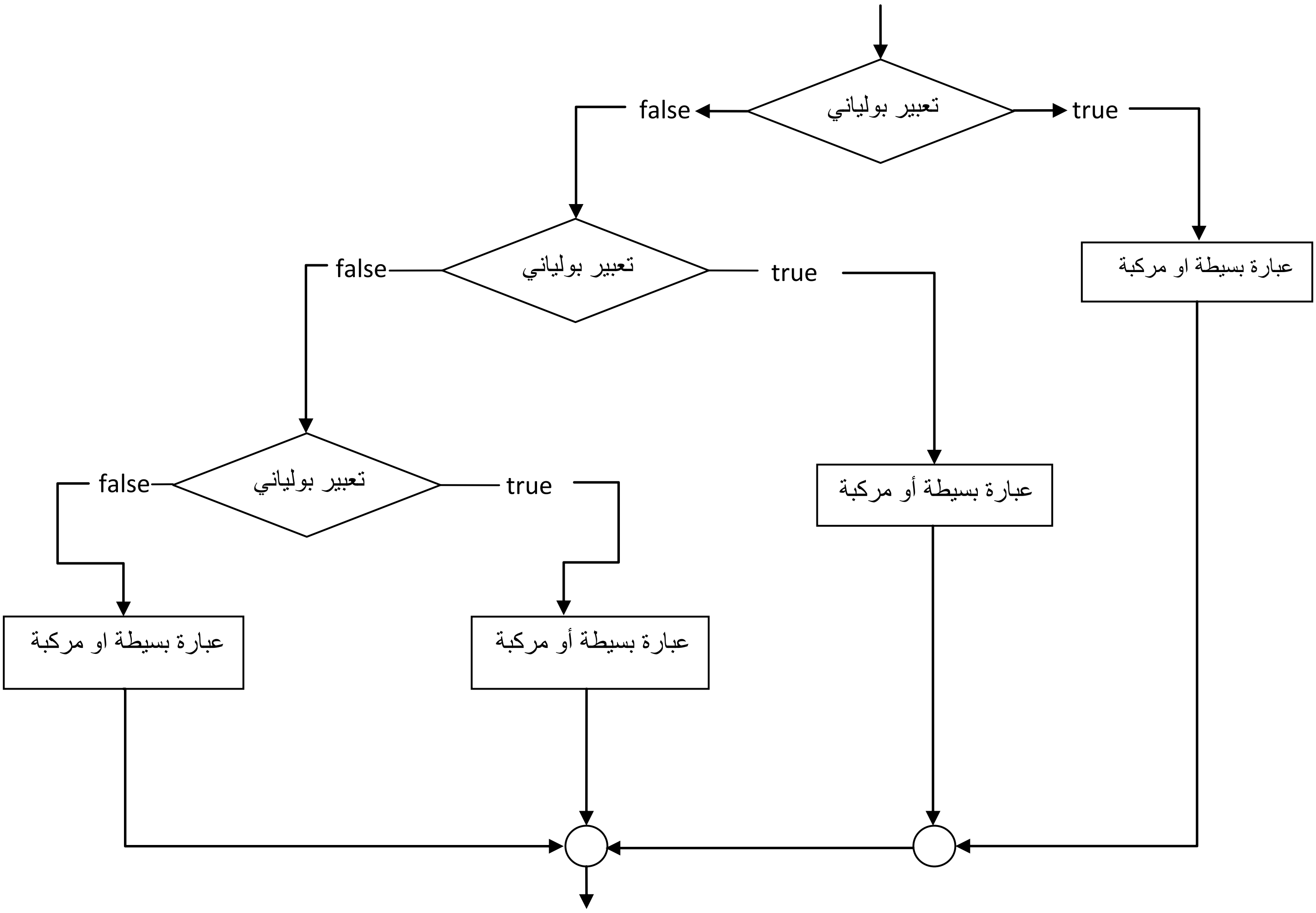
```
value? 13
val =       40; odd=       0; even =        1
val =       20; odd=       0; even =        2
val =       10; odd=       0; even =        3
val =        5; odd=       1; even =        3
val =       16; odd=       1; even =        4
val =        8; odd=       1; even =        5
val =        4; odd=       1; even =        6
val =        2; odd=       1; even =        7
val =        1; odd=       2; even =        7
After        9 iterations, result = 1.
       7 even values;        2 odd values
```

المهندس خالد ياسين الشيخ

خرج البرنامج السابق من أجل val=8 هو:

```
value? 8
val =        4; odd=       0; even =       1
val =        2; odd=       0; even =       2
val =        1; odd=       1; even =       2
After        3 iterations, result = 1.
       2 even values;        1 odd values
```

نلاحظ من خلال الخرج السابق أن البرنامج يحتاج إلى 3 خطوات حتى تصل الخوارزمية إلى الخطوة 2 وهي val=1.
و يحتاج البرنامج السابق من أجل val=13 إلى 9 خطوات حتي يصل إلى val=1.
و الشكل التالي يُرينا المخطط الانسيابي أو المنطقي=التخيلي=الافتراضي لعبارة if-else الموسعة



مثال:نريد مثلاً أن نكتب سلسلة معينة من الاعداد و التي تقبل القسمة على أعداد أولية مثل :2 and 3 and 5 and 7 فإذا قبل العدد المفحوص القسمة على أحد هذه الإعداد فسيكتب العدد و سوف ينتقل الفحص إلى عدد تالي له و البرنامج التالي يُرينا كيفية تحقيق ذلك:

Code

```
program test;
const prime1=2;
     prime2=3;
     prime3=5;
     prime4=7;
var count,maxtrials,startvalue,result:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
begin
getinteger('starting value?',startvalue);
getinteger('maximum value? (> startvalue)',maxtrials);
for count:=startvalue to maxtrials do
begin
write(count:3,': ');
if count mod prime1=0 then
write(prime1:4)
else
if count mod prime2=0 then
write(prime2:4)
else
if(count mod prime3=0) then
write(prime3:4)
else
if(count mod prime4=0)then
write(prime4:4)
else
write('nil');
writeln;
end;
readln;
end.
```

خرج البرنامج السابق هو (من 2 و حتى 30) هو:

```
starting value? 2
maximum value? (> startvalue) 30
  2:    2
  3:    3
  4:    2
  5:    5
  6:    2
  7:    7
  8:    2
  9:    3
 10:    2
 11: nil
 12:    2
 13: nil
 14:    2
 15:    3
 16:    2
 17: nil
 18:    2
 19: nil
 20:    2
 21:    3
 22:    2
 23: nil
 24:    2
 25:    5
 26:    2
 27:    3
 28:    2
 29: nil
 30:    2
```

و حتى نحصل على كل القواسم الأولية للعدد يجب أن نُحول عبارة if-else السابقة إلى عبارات if المستقلة كما يبين البرنامج التالي:

Code

```
program test;
const prime1=2;
    prime2=3;
    prime3=5;
    prime4=7;
var count,maxtrials,startvalue,result:integer;
ok:boolean;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
begin
getinteger('starting value?',startvalue);
getinteger('maximum value? (> startvalue)',maxtrials);
for count:=startvalue to maxtrials do
begin
ok:=true;
write(count:3,': ');
if count mod prime1=0 then
begin
write(prime1:4);
ok:=false;
end;
```

```
if count mod prime2=0 then
begin
write(prime2:4);
ok:=false;
end;
if(count mod prime3=0) then
begin
write(prime3:4);
ok:=false;
end;
if(count mod prime4=0)then
begin
write(prime4:4);
ok:=false;
end;
if ok then
write('nil');
writeln;
end;
readln;
end.
```

```
starting value? 2
maximum value? (> startvalue) 30
  2:    2
  3:    3
  4:    2
  5:    5
  6:    2   3
  7:    7
  8:    2
  9:    3
 10:    2   5
 11: nil
 12:    2   3
 13: nil
 14:    2   7
 15:    3   5
 16:    2
 17: nil
 18:    2   3
 19: nil
 20:    2   5
 21:    3   7
 22:    2
 23: nil
 24:    2   3
 25:    5
 26:    2
 27:    3
 28:    2   7
 29: nil
 30:    2   3   5
```

الاختيار ما بين عدة إمكانيات – عبارة case

selection from among several possibilities :the case statement

تملك لغة باسكال نوعاً آخر من بنى الاختيار و هو نوع مفيد جداً عندما يكون ناتج الفحص قيمة عددية أو عندما يكون لنتيجة الفحص احتمالات متعددة فمثلاً عندما نريد أن نفحص حرفاً فيما إذا كان حرفاً صوتياً و ذلك باستخدام عبارة if سيكون الفحص طويلاً كالتالي:

```
If (testch='a') or (testch='e') or (testch='i') or (testch='o') or (testch='o')
or (testch='u') or (testch='u') or(testch='A') or(testch='E') or (testch='I)
or (testch='U')  then ..................
```

أو افترض=تخيل أننا نفحص رمزاً ما فيما إذا كان حرفاً صوتياً vowel أو حرفاً ساكناً consonant أو رقماً زوجياً even digit أو رقماً فردياً odd digit أو رمزاً فإننا قد نفكر باستخدام عبارة if-else الموسعة و التي ستاخذ البنية التالية:

```
If testch  حرف ساكن  then...
              Else if testch  حرف صوتي then ...
                     Else if testch  رقم زوجي then....
                            Else if testch رقم زوجي then...
                                   Else........
```

كل عملية فحص من العمليات السابقة (مثل: هل testch هو حرف ساكن أو هل testch رقم فردي) يمكن أن يتألف من تعابير بولينية طويلة و حتى نتخلص من هذه التعابير الطويلة أعطتنا لغة باسكال عبارة case التي سهلت عملية الفحص تلك أما لغة التربو باسكال فقد جعلت الأمر أكثر سهولة و أيسر تعاملاً بإضافة ميزات لهذه العبارة كما سنرى لاحقاً.
تزودنا عبارة case بإمكانية الاختيار اعتماداً على قيمة المتحول أو التعبير و التي يجب أن ينتمي إلى أحد أنواع المعطيات المرتبة ordinal type و تسمح لنا عبارة case و بشكل أساسي أن نجمع و نرتب كل قيم المتحول أو التعبير التي تقودنا إلى عمل معين.
يفحص البرنامج التالي رمزاً معيناًمن حيث انتمائه إلى إحدى المجموعات التي ذكرناها قبل قليل (حرف صوتي- حرف ساكن- رقم زوجي- رقم آخر- رمز آخر).

Code

```
program test;
var teststr:string;
index,howlong,nrconsonants,nrvowels,
nreven,nrodd,nrprintable:integer;
ch:char;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
begin
getstring('string to process?',teststr);
howlong:=length(teststr);
nrconsonants:=0;
nreven:=0;
nrodd:=0;
nrprintable:=0;

for index:=1 to howlong do
begin
ch:=upcase(teststr[index]);
case ch of
```

```
'A','E','I','O','U':inc(nrvowels);
'B'..'D','F'..'H','J'..'N','P'..'T','V'..'Z':inc(nrconsonants);
'0','2','4','6','8':inc(nreven);
'1','3','5','7','9':inc(nrodd)
else
inc(nrprintable);
end;
end;
writeln('Nrvowels    =',nrvowels:3,
            ';nrconsonants  =',nrconsonants:3);
writeln('nreven      =',nreven:3,
            '; nrodd       =',nrodd:3);
writeln('nrprintable = ',nrprintable:2);
readln;
end.
```

خرج البرنامج من أجل العبارة 'syrian arab republic25-8-1983'

```
string to process? syria arab republic25-8-1983
Nrvowels     =  7;nrconsonants  = 10
nreven       =  3; nrodd        =  4
nrprintable =  4
```

المهندس خالد ياسين الشيخ

يقرأ البرنامج السابق سلسلة رمزية و يفحص كل رمز فيها معتمداً على الفئة التي ينتمي إليها هذا الرمز و يتم زيادة عداد الفئة التي ينتمي إلليها هذا الرمز و تم استخدام التابع length مسبق التعريف في لغة تربو باسكال حيث يعيد هذا التابع عدد الرموز المكونة للسلسلة الرمزية الممررة إليه و يحدد التركيب (teststr[index] الرمز الذي رقمه index ضمن السلسلة teststr و التابع Upcase يحول الحرف المرر إلى حرف كبير.

نشرح الآن كيف تعمل عبارة case السابقة:

- تفحص قيمة ch أولا عند كل تنفيذ لعبارة case .
- إذا كانت قيمة ch عبارة عن حرف صوتي أي ينتمي إلى أحد الأحرف الخمسة السابقة فإن عداد هذه الفئة سيزداد بمقدار 1 أي أن العبارة بسيطة كانت أم مركبة و التي تأتي بعد النقطتان : سو تنفذ في حال تطابق قيمة ch مع القيم المذكورة قبل هاتين النقطتين و يتابع البرنامج مع التكرار وفق حلقة for و إلا فسيتابع سيره و يقارن قيمة ch مع القيم التالية.
- إذا لم تكن القيمة حرفاً صوتياً فإن البرنامج سوف يقارنها مع المجموعة الثانية من القيم و التي تمثل في مثالنا الأحرف الساكنة و قد مثلناها هما على شكل مجموعات جزئية فكل الأحرف بين الحرفين B و D (التي هي B و C و D) و كذلك الأحرف بين J و N و هكذا... و لتمثيل هذا المجال من الرموز وضعنا نقطتين متتاليتين بين القيمة البدائية و القيمة النهائية فإذا كانت قيمة ch أحد هذه الحروف فإذا كانت قيمة ch أحد هذه الحروف فإن عداد الفئة nrconsonants سوف يزداد بمقدار 1 و تنتهي عبارات case و يتابع البرنامج التكرار وفق حلقة for.
- إذا لم يكن الرمز حرفاً صوتياً أو حرفاً ساكناً فإن البرنامج سيفحص كونه رقماً زوجياً أو رقماً فردياً.
- إذا لم يتصنف الرمز عند هذه النقطة وفق أي مجموعة من المجموعات السابقة فإن عبارة else ستنفذ أي أن العداد nrprintable سيزداد بمقدار 1.

ومما يجب ملاحظته هنا أننا لا نضع نقطتين بين else و العبارة الواجب تنفيذها و إنما أنهينا عبارة case بواسطة end التي ليس لها begin و هي من إحدى المواضع الثلاثة في لغة التربو باسكال التي يجب علينا فيها أن نضع end بدون begin لها (المواضع هي تعريف السجل و تعريف الكائن object و تعريف عبارة case).

و الصيغة الكتابية لعبارة case هي كما يوضح الجدول أدناه:

```
Case <متحول ينتمي إلى أحد أنواع المعطيات المرتبة أو تعبير> of
        <لائحة من القيم>:<عبارة بسيطة أو مركبة>
        <لائحة من القيم>:< عبارة بسيطة أو مركبة>
        ............
        <لائحة من القيم>:<عبارة بسيطة أو مركبة>
```

```
        Else <عبارة بسيطة أو مركبة>
    End;
```

أن السطر الذي يحوي else هو سطر اختياري في لغة التربو باسكال (أما في لغة باسكال القياسية فهذا السطر غير مسموح).
يجري التحكم بعبارة case وفق قيمة المتحول أو قيمة التعبير الذي يأتي مباشرة بعد المميز المحجوز case و هذه القيمة يمكن أن تكون إحدى القيم الموجودة ضمن لوئح القيم في عبارة case و في حال وجود تعبير بعد المميز المحجوز case مثل myint+7 فسُحسب هذا التعبير أولاً و على ضوء ناتج الحساب ستنفذ عبارة case .
يمكن أن تتألف لأئحة القيم value list من قيمة واحدة أو أكثر مفصولة عن بعضها بفاصلة (,) و يمكن أت تكون مجالاً جزئياً من القيم محدداً بقيمة البداية و يأتي بعد نقطتان متتاليتان (..) ومن ثم قيمته النهائية و هذه الإمكانية (أي أن تشمل لائحة القيم على مجالات جزئية)هي توسيع ة تطوير للغة باسكال القياسية و هي متوفرة فقط في لغة التربو باسكال).
عندما يصادف البرنامج عبارة case فإنه يأخذ قيمة المتحول أو التعبير و يبحث عن القيمة الطابقة ضمن لوائح القيم حتى يجد القيمة المطابقة وعندئذ يُنفذ العبارة المرتبطة بهذه اللائحة و تنتهي عبارة case مباشرة أما إذا لم يجد أي قيمة مطابقة فإنه ينفذ العبارة المرتبطة بالجزء else في حال وجودها و تنتهي عبارة case أما إذا لم يجد المميز else فإن البرنامج لن يعمل شيئاً داخل عبارة case و يتابع البرنامج تنفيذ عباراته ابتداء من العبارة التي تأتي مباشرة بعد عبارة case .
يمكن أن تكون العبارة المرتبطة بلائحة القيم عبارة بسيطة أو مركبة و في الحقيقة يمكن أن يكون هذه العبارة عبارة case أخرى أي يمكن بناء عبلوات case متداخلة فيما بينها و البرنامج التالي يوضح لنا استخدام عبارة case المتداخلة :

Code

```
program test;

const maxtrials=10;
var count,value:integer;
begin
randomize;
for count:=1 to maxtrials do
begin
value:=random(maxint);
case value mod 4 of
0:begin
writeln(value:5,' is divisible by 4');
case value mod 2 of
0:writeln(value:15,' is even');
1:writeln(value:15,' is odd');
end;
end;
1:writeln(value:5,': remainder is 1');
2:writeln(value:5,': remainder is 2');
3:writeln(value:5,': remainder is 3');
end;
end;
readln;
end.
```

خرج البرنامج السابق من أجل تنفيذ عشوائي مشابهاً للتالي:

```
23942: remainder is 2
29099: remainder is 3
12234: remainder is 2
15412 is divisible by 4
          15412 is even
14838: remainder is 2
27086: remainder is 2
 3987: remainder is 3
18830: remainder is 2
 6960 is divisible by 4
           6960 is even
30741: remainder is 1
```

يُولد البرنامج السابق عشرة أعداد صحيحة عشوائية و يعطينا بعض المعلومات حول هذه الأعداد فإذا كان العدد العشوائي يقبل القسمة على 4 فإن البرنامج يُولد عدداً عشوائياً آخر ويفحص هذا العدد فيما إذا كان عدداً فردياً أم زوجياً و عملية الفحص هذه تنجزها عبارة case المتداخلة مع عبارة case الخارجية و تُنهى عبارة case المتداخلة بواسطة end الأولى التي تأتي بعد الحالة 1 وأما end الثانية فهي لإنهاء العبارة المركبة المرتبطة بالحالة 0.
لاحظ أن عبارتي Case في البرنامج السابق يتم التحكم بهما بواسطة تعبيرين قيمهما المتوقعة تقع ضمن مجالات صغيرة من الأعداد الصحيحة(0,1,2,3) و (0,1).

بنية التكرار المتهورة – عبارة rpeat : impulsive iteration construct –the repeat statement:

يحاول البرنامج التالي الحصول على عدد عشوائي يقبل القسمة على 17 .

Code

```
program test;
const divisor=17;
maxtrials=1000;
var value,count,nrtries,runningsum,
shortestrun,longestrun:longint;
procedure checkexterms(value:longint; var shortest,longest:longint);
begin
if value>longest then
longest:=value;
if(value<shortest) then
shortest:=value;
end;
procedure displaysummary(nrtrials,total,shortest,longest:longint);
begin
writeln('longest run  = ',longest:3);
writeln('shortest run = ',shortest:3);
writeln('Average run length = ',total/nrtrials:10:3);
end;
begin
randomize;
runningsum:=0;
shortestrun:=maxint;
longestrun:=0;
writeln('Geenerating values...');
for count:=1 to maxtrials do
begin
```

```
nrtries:=0;
repeat
value:=random(maxint);
inc(nrtries);
until (value mod divisor=0);
checkexterms(nrtries,shortestrun,longestrun);
inc(runningsum,nrtries);
end;
displaysummary(maxtrials,runningsum,shortestrun,longestrun);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي (من أجل تنفيذ عدة مرات):

```
Geenerating values...
longest run  = 113
shortest run =   1
Average run length =    17.798
```

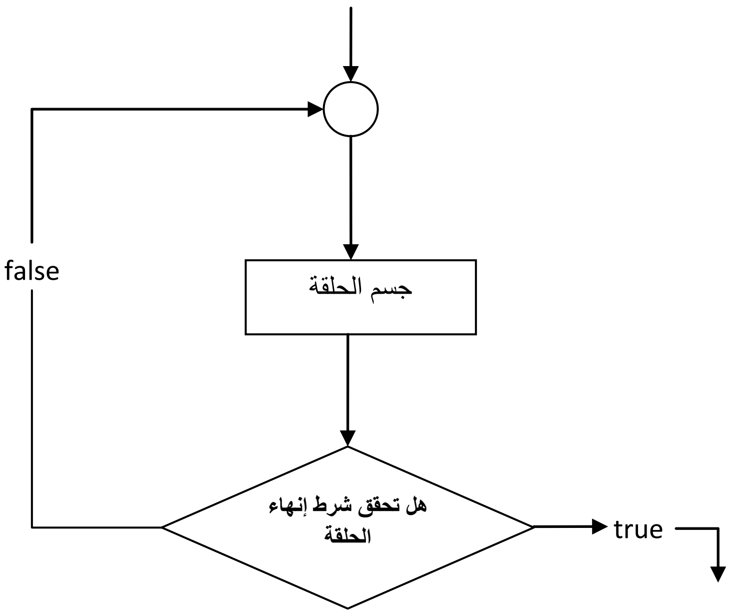
```
Geenerating values...
longest run  = 171
shortest run =   1
Average run length =    17.092
```

```
Geenerating values...
longest run  = 123
shortest run =   1
Average run length =    17.201
```

لقد وضعنا في البرنامج السابق حلقة repeat بشكل متداخل مع حلقة for و حلقة repeat تكرر تنفيذ مجموعة من العبارات حتى تصبح قيمة شرط معين true و هذا الشرط يدعى شرط الإنهاء termination condition إذ أن تنفيذ الحلقة حتى تصبح قيمة شرط الإنهاء true .
إذا غيرنا قيمة الثابت maxtrials إلى 100 و نفذ البرنامج عدة مرات ستلاحظ تغييراً جذرياً عن النتائج السابقة (ابتهاد مجال القيم عن القيمة الوسطى) لأنه كلما كان عدد المحاولات أقل فإن النتائج ستكون أقل استقراراً.
قم بغيير قيمة المقسوم عليه divisor واعد تنفيذ البرنامج السابق ستلاحظ أن النتائج ستتجمع حول مجال قيم مختلفة حيث أن معدل طول التنفيذ average run يتعلق مباشرة بالمقسوم عليه.
فمثلاً إذا كان المقسوم عليه 2 يمكن أن نتوقع الحصول على عدد زوجي في كل عدد عشوائي مولد ولذلك ستكون القيم العظمى و الصغرى أصغر منه إذا كبرنا المقسوم عليه.
و الصيغة الكتابية لحلقة repeat هي التالية:

```
repeat
        <تتابع من العبارات البسيطة أو المركبة>
UNTIL <تعبير بولياني>
```

التعبير البولياتي في عبارة REPEAT هو نفسه شرط الانهاء و مما يجب ملاحظته أننا لم نستخدم المميزين المحجوزين begin و end لتحديد بداية و نهاية هذه العبارة إذ يحدد المميز repeat بداية هذه الحلقة و يحدد نهايتها المميز المحجوز until و حلقة repeat تنفذ عباراتها مرة واحدة أولاً ومن ثم تفحص فيما إذا كانت ستتوقف أو تتكرر مرة أخرى.(إن شكل التنفيذ هذا يكون التنفيذ ثم الفحص) يعني أن الحلقة ستنفذ مرة واحدة على الأقل و من ثم تأتي عملية الفحص و من هنا جاءت تسمية هذه العبارة ببنية التكرار المتهورة و المخطط الانسيابي أو المنطقي=الافتراضي=التخيلي لهذه الحلقة هو التالي:



يستخدم في شرط الإنهاء عادة متحول تحدد قيمته ناتج التعبير البولياني فيما إذا كان true أو False و هذا المتحول يدعى عادة متحول الحلقة loop variable و تغير قيمة المتحول هذا حسب الحاجة أثناء عمليات التكرار لأن قيمته هي التي تحدد متى تنتهي الحلقة. أما إذا بقيت قيمة هذا المتحول ثابتة و لم تستطع أن تجعل قيمة شرط الإنهاء true فلن يتوقف تنفيذ الحلقة أبداً.

تختلف حلقة repeat عن حلقة for بأنه علينا تغير متحول الحلقة أو على الأقل أن نخبر البرنامج كيف يقوم بذلك عوضاً عنا في حين أنه في حلقة for يتغير متحول الحلقة تلقائياً بعد كل تنفيذ للحلقة.

و حلقة repeat تنفذ مرة واحدة فقط و من ثم يفحص شرط الإنهاء وهذا يعني أن الحلقة تنفذ مرة واحدة فقط على الأقل و يجب أن لا ننسى ايضاً ان تغير متحول الحلقة هي مسؤوليتنا و علينا التأكد من ان شرط الإنهاء سوف يأخذ القيمة true بعد عدد معين من مرات التنفيذ قل هذا العدد أو كثر (شرط الإنهاء يمكن أن يكون بسيط أو مركب).

ليكن لدينا البرنامج التالي:

Code

```
program test;
var i:integer;
begin
randomize;
repeat
i:=random(5);
until(i mod 10=0);
writeln(i);
readln;
end.
```

خرج البرنامج السابق هو:

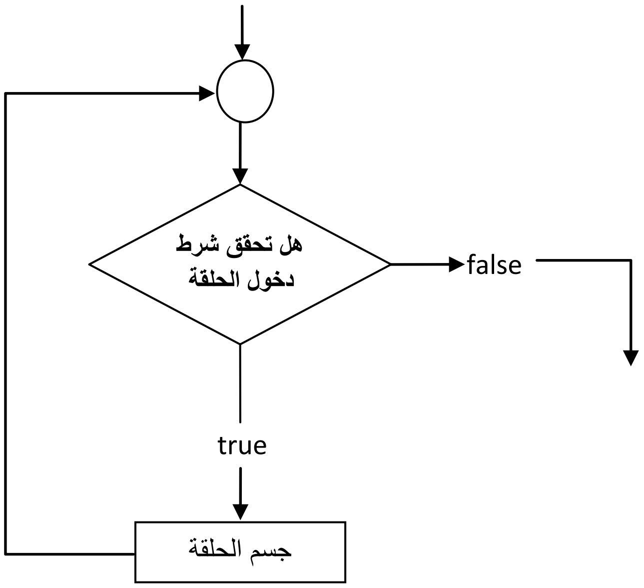
A. **0**
B. لا يمكن معرفة خرج البرنامج
C. 4
D. تسبب حلقة Repeat الدخول بحلقة لا نهائية
E. None of the above

بنية التكرار الحذرة – عبارة while - the cautious iteration construct –the while statement:

تفحص حلقة while شرط الاستمرار أولاً ثم تنفذ جسم الحلقة على عكس حلقة repeat التي تنفذ تعليماتها ومن ثم تفحص شرط الإنهاء و الصيغة الكتابية لحلقة while هي:

While <تعبير بولياني> <عبارة بسيطة أو مركبة>

يُدعى التعبير البولياني في هذه الحلقة بشرط الاستمرار continuation condition إذ إن تنفيذ هذه الحلقة يستمر طالما أن شرط الاستمرار محقق أي قيمته true و يُنهى تنفيذ هذه الحلقة عندما تصبح قيمة هذا الشرط false على عكس حلقة repeat التي ينتهي تنفيذها عندما تصبح قيمة شرط الإنهاء true و المخطط الانسيابي أو المنطقي=الافتراضي= التخيلي لهذه الحلقة هو:



لنأخذ مثالاً للبرنامج الذي يبحث عن عدد يقبل القسمة على 17 و لكن باستخدام حلقة while:

Code

```
program test;
const divisor=17;
maxtrials=1000;
var value, count, nrtries,
  runningsum,shortestrun,longestrun:longint;
procedure checkextremes(value:longint; var shortest,longest:longint);
begin
if(value>longest)then
longest:=value;
if(value<shortest)then
shortest:=value;
end;
```

```
procedure displaysummary(nrtrials,total,shortest,longest:longint);
begin
writeln('longest run  = ',longest:3);
writeln('shortest run = ',shortest:3);
writeln('average run length = ',total/nrtrials:10:3);
end;
begin
randomize;
runningsum:=0;
shortestrun:=maxint;
longestrun:=0;
writeln('Geenerating value...');
for count:=1 to maxtrials do
begin
value:=random(maxint);
nrtries:=1;
while(value mod divisor<>0)do
begin
value:=random(maxint);
inc(nrtries);
end;
checkextremes(nrtries,shortestrun,longestrun);
inc(runningsum,nrtries);
end;
displaysummary(maxtrials,runningsum,shortestrun,longestrun);
readln;
end.
```

```
Geenerating value...
longest run  = 115
shortest run =   1
average run length =     16.020
```

```
Geenerating value...
longest run  = 134
shortest run =   1
average run length =     16.653
```

خرج البرنامج السابق شبيه بالتالي (تنفيذ البرنامج عدة مرات):
أن لهذا للبرنامج نفس وظيفة البرنامج السابق باستثناء أن الحلقة الداخلية بُنيت بواسطة حلقة while عوضاً عن حلقة repeat و هذا التبديل قادنا إلى مجموعة من التبديلات هي:

1. تفحص حلقة while الشرط قبل تنفيذ أي شيء لذلك كان من الضروري أن تتولد القيمة العشوائية الصحيحة خارج الحلقة و تمرر هذه القيمة إلى شرط الاستمرار في حلقة while (ينبغي تخيئة متحول الحلقة خارج الحلقة أما في حلقة repeat فإن عملية تهيئة متحوّل الحلقة يمكن أن تقع ضمن جسم الحلقة.

2. بما اننا نبحث عن القيمة false للتعبير البولياني في الحلقة while أي أننا نبحث عن قيمة لا تقبل القسمة على المقسوم عليه divisor فإن عملية الفحص هذه ستكون معاكسة تماماً لعملية الفحص في حلقة repeat.

يمكننا بناء حلقات متكافئة تماماً بواسطة while أو repeaat أي أن هذه البنى متكافئة بمعنى أنه استخدام حلقة مكان الحلقة الأخرى.

إذا كانت قيمة شرط الاستمرار false في بداية التنفيذ في عبارة while فإن حلقة while لن تنفذ أبداً و تطبق حلقة while على العبارة الملحقة بها بسيطة كانت أو مركبة وعند استخدان بنى مركبة يجب أن تُحاط هذه العبارة بالمميزين begin و end لتحديد بداية ونهاية هذه العبارة و كما هو الحال في عبارة repeat فإن تغيير متحول الحلقة لا يتم بشكل تلقائي وإنما تغيير قيمته بواسطة إلحاق ضمن الحلقة أو استدعاء لإجراء أو تابع.

و حلقة while تفحص شرط الاستمرار قبل الدخول في جسم الحلقة ومن هنا جاءت تسميتها تجاوزاً بالحلقة الحذرة فإذا تحقق شرط الاستمرار ينفذ جسم الحلقة الذي يجب أن يتضمن عملية تغيير لقيمة متحول الحلقة أما إذا لم يتحقق الشرط فلن تنفذ هذه الحلقة.

و البرنامج التالي يُظهر الرموز القابلة للطباعة PRINTABLE موضحاً من خلال ذلك أهمية تغيير قيمة متحول الحلقة:

Code

```
program test;
var count:integer;
   currch:char;
procedure display;
begin
write('   ',ord(currch):3,' ',currch);
inc(count);
if(count mod 5 =0)then
writeln;
end;
begin
count:=0;
currch:=#32;
while(ord(currch)<127)do
begin
display;
inc(currch);
end;
readln;
end.
```

خرج البرنامج السابق هو:

```
 32         33  !    34  "    35  #    36  $
 37  %      38  &    39  '    40  (    41  )
 42  *      43  +    44  ,    45  -    46  .
 47  /      48  0    49  1    50  2    51  3
 52  4      53  5    54  6    55  7    56  8
 57  9      58  :    59  ;    60  <    61  =
 62  >      63  ?    64  @    65  A    66  B
 67  C      68  D    69  E    70  F    71  G
 72  H      73  I    74  J    75  K    76  L
 77  M      78  N    79  O    80  P    81  Q
 82  R      83  S    84  T    85  U    86  V
 87  W      88  X    89  Y    90  Z    91  [
 92  \      93  ]    94  ^    95  _    96  `
 97  a      98  b    99  c   100  d   101  e
102  f     103  g   104  h   105  i   106  j
107  k     108  l   109  m   110  n   111  o
112  p     113  q   114  r   115  s   116  t
117  u     118  v   119  w   120  x   121  y
122  z     123  {   124  |   125  }   126  ~
```

يُنجز البرنامج السابق مهمتين طالما أن قيمة المتحول currch اقل من 127 إذ يُظهر الرمز و يزيد قيمة المتحول currch ضمن الحلقة.

إذا حذفنا السطر التالي ضمن البرنامج السابق:

inc(currch);

فإن البرنامج لن ينتهي و سيُظهر رمز المسافة الذي شفرة ASCII المقابلة له هي 32 بشكل لا نهائي (تدخل حلقة while في حلقة لا نهائية بسبب عدم وجود شرط مناسب لإنهاء الحلقة).

بنية عبارات التكرار the structure of iteration statements:

يمكن أن يكون لعبارات التكرار تأثيرات جانبية على الرغم من بساطة صياغتها الكتابية إذ يمكن أن يختلف خرج الحلقتين repeat و while اختلافاً بسيطاً اعتماداً على طريقة التعامل مع التعبير البولياني الذي يتحكم بالحلقة و في الحقيقة هناك ثلاثة شروط تؤثر على عمل حلقات التكرار هي:

1. القيمة الابتدائية و النهائية لمتحول الحلقة.
2. طريقة الفحص الفعلية للتعبير البولياني فيما إذا كان التعبير البولياني شرط استمرار أو شرط إنهاء.
3. مكان تغيير متحول الحلقة ضمن جسم الحلقة.

و البرنامج التالي يوضح تأثير العوامل الثلاثة السابقة على الحلقات .إذ يحسب البرنامج و يطبع بعض القيم العددية

Code

```
program test;
const maxvalue=10;
function cube(val:integer):integer;
begin
cube:=val*val*val;
end;
procedure wchangeAtTop;
var count,runningsum,result:integer;
begin
count:=1;
runningsum:=0;
while(count<=maxvalue)do
begin
inc(count);
result:=cube(count);
inc(runningsum,result);
writeln(count:2,': cube = ',result:4,'; runningsum = ',runningsum:4);
end;
end;
procedure wchangeAtBottom;
var count, runningsum, result:integer;
begin
count:=1;
runningsum:=0;
while(count<=maxvalue)do
begin
result:=cube(count);
inc(runningsum,result);
writeln(count:2,': cube = ',result:4,'; runningsum = ',runningsum:4);
inc(count);
end;
end;
begin
writeln('While ChangeAtTop:');
WchangeAtTop;
writeln('---------------------------------');
```

```
writeln('While ChangeAtBottop:');
WchangeAtbottom;
readln
end.
```

خرج البرنامج السابق هو كالتالي:

```
While ChangeAtTop:
 2: cube =    8; runningsum =    8
 3: cube =   27; runningsum =   35
 4: cube =   64; runningsum =   99
 5: cube =  125; runningsum =  224
 6: cube =  216; runningsum =  440
 7: cube =  343; runningsum =  783
 8: cube =  512; runningsum = 1295
 9: cube =  729; runningsum = 2024
10: cube = 1000; runningsum = 3024
11: cube = 1331; runningsum = 4355
----------------------------------
While ChangeAtBottop:
 1: cube =    1; runningsum =    1
 2: cube =    8; runningsum =    9
 3: cube =   27; runningsum =   36
 4: cube =   64; runningsum =  100
 5: cube =  125; runningsum =  225
 6: cube =  216; runningsum =  441
 7: cube =  343; runningsum =  784
 8: cube =  512; runningsum = 1296
 9: cube =  729; runningsum = 2025
10: cube = 1000; runningsum = 3025
```


الفرق الوحيد بين الإجراءين wchangeAttop و WchangeAtbottom هو في موقع العبارة التالية:

inc(count);

ففي الإجراء wchangeAttop كانت قيمة متحول الحلقة تتغير قبل عمل أي شيء ضمن الحلقة و في هذه الحالة فإن الحلقة سوف تعمل من القيمة 2 و حتى 11 عوضاً عن عملها من القيمة 1 و حتى 10 أما عند وضع هذا التعبير لمتحول الحلقة في نهاية جسم الحلقة كما هو الحال في الإجراء wchangeAtBottom فإن هذه القيمة سوف تتغير من 1 و حتى 10 .

يمكننا استخدام حلقة repeat ضمن الإجراءات السابقة عوضاً عن حلقة while مع الانتباه إلى أن شرط الإنهاء termination condition المقابل لشرط الاستمرار continuation condition في البرنامج السابق سيكون:

Count>maxvalue

مثال للمقارنة A comparison Example:

المثال التالي يمكننا من مقارنة سلوك الحلقتين while و repeat مباشرة إذ يُرينا ناتج عملية الفحص قبل أو بعد الدخول في جسم الحلقة:

Code

```
program test;
var value:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
procedure whiletest(count:integer);
begin
while(odd(count))do
begin
write(count,' : ');
count:=count div 2;
end;
writeln('Done');
```

```
end;
procedure repeattest(count:integer);
begin
repeat
write(count,' : ');
count:=count div 2;
until(not(odd(count)));
writeln('Done');
end;
begin
repeat
Getinteger('value ? (0 to stop) ',value);
if(value<>0)then
begin
writeln('whiletest:');
whiletest(value);
writeln('-----------------');
writeln('repeattest:');
repeattest(value);
end;
until(value=0);
readln;
end.
```

تمثل الأسطر التالية عينية من خرج البرنامج السابق من أجل قيم ابتدائية للمتحول value يختارها المستخدم user لاحظ سلوك الحلقتين من اجل هذه القيم؟؟.

```
value ? (0 to stop)  37
whiletest:
37 : Done
-----------------
repeattest:
37 : Done
value ? (0 to stop)  14
whiletest:
Done
-----------------
repeattest:
14 : 7 : 3 : 1 : Done
value ? (0 to stop)  201
whiletest:
201 : Done
-----------------
repeattest:
201 : Done
value ? (0 to stop)  36
whiletest:
Done
-----------------
repeattest:
36 : Done
value ? (0 to stop)  29
whiletest:
29 : Done
-----------------
repeattest:
29 : Done
value ? (0 to stop)  31
whiletest:
31 : 15 : 7 : 3 : 1 : Done
-----------------
repeattest:
31 : 15 : 7 : 3 : 1 : Done
value ? (0 to stop)  0
```

المهندس خالد ياسين الشيخ

لاحظ من أجل الخرج السابق أنه من أجل قيم معينة قد لا تعطينا حلقة while أي نتيجة في حين أن حلقة repeat دائماً تعطينا و على الأقل قيمة واحدة.

تجاوز قوانين الحلقات Getting Around the looping rules:

تملك لغة التربو باسكال روتينات مسبقة التعريف تمكننا من القيام بأعمال خاصة أثناء تنفيذ الحلقات و هذه الإجراءات هي break و continue و exit و halt ووظيفة هذه الإجراءات تغيير انسياب التحكم flow of control ضمن البرنامج ومن هذه الإجراءات ما هو مخصص للحلقات for و repeat و while فلا تستخدم إلا ضمن هذه الحلقات و هذه الإجراءات هي continue و break و منها ما هو عام يمكن استخدامه في أي موقع من البرنامج .
يستخدم الإجراءان continue و break دائماً ضمن العبارات المركبة المرتبطة بالبنى الحلقية أو البسيطة.

الإجراء continue:

و هو أقل هذه الإجراءات تمزيقاً للحلقة إذ يسبب هذا الإجراء بداية تنفيذ التكرار التالي للحلقة مباشرة و العبارات التالية لاستدعاء هذا الإجراء سوف تهمل و التكرار الحالي ينتهي و يبدأ التكرار التالي بتنفيذ عباراته بشكل طبيعي طبعاً ما لم يستدع الإجراء continueمرة أخرى.
لنفترض أننا نريد أن نكتب برنامجاً يُظهر كل القيم بين 1 و 20000 و التي تقبل القسمة على الأعداد الخمسة الأولية التالية 11,7,5,3,2, و خوارزمية هذا البرنامج تعتمد على عملية فحص قابلية القسمة على كل عدد من هذه الأعداد الأولية و بالتتالي و لكن البرنامج سيكون أسرع فيما لو لم نكمل عملية الفحص في حال فشل إحدى الفحوصات على العدد المحدد.
و البرنامج التالي يستخدم الإجراء continue لتسريع خوارزمية الحل:

Code

```
program test;
const target=20000;
var count:integer;
procedure continuetest;
begin
count:=0;
while(count<=target)do
begin
inc(count);
if(count mod 2 <>0)then
continue
else
if(count mod 3<>0)then
continue
else
if(count mod 5<>0)then
continue
else
if(count mod 7<>0)then
continue
else
if(count mod 11<>0)then
continue
else
writeln(count);
end;
writeln('upon exit, count = ',count);
end;
begin
continuetest;
writeln('press to enter to exit');
readln;
end.
```

خرج البرنامج السابق هو:

```
2310
4620
6930
9240
11550
13860
16170
18480
upon exit, count = 20001
press to enter to exit
```

والبرنامج التالي يوضح لنا عمل الإجراء continue :

Code
program test; const target=20; var i:integer; begin for i:=1 to target do begin if (i mod 4=0) or (i mod 3=1)then continue; write(i,','); end; readln; end.

خرج البرنامج السابق:

```
2,3,5,6,9,11,14,15,17,18,
```

الإجراء Break :

ينهي الإجراء break تنفيذ الحلقة مباشرة عند استدعائه و ينقل التحكم إلى أول عبارة تلي الحلقة و حتى نرى الفرق بين الإجراءين continue و break لنغير كل استدعاءات الإجراء continue في البرنامجين السابقين بالإجراء break و نلاحظ الفرق :

Code
program test; const target=20000; var count:integer; procedure breaktest; begin count:=0; while(count<=target)do begin inc(count); if(count mod 2 <>0)then break

```
else
if(count mod 3<>0)then
break
else
if(count mod 5<>0)then
break
else
if(count mod 7<>0)then
break
else
if(count mod 11<>0)then
break
else
writeln(count);
end;
writeln('upon exit, count = ',count);
end;
begin
breaktest;
writeln('press to enter to exit');
readln;
end.
```

خرج البرنامج السابق هو:

```
upon exit, count = 1
press to enter to exit
```

و البرنامج التالي يوضح عمل الإجراء break:

Code

```
program test;
const target=20;
var i:integer;
begin
for i:=1 to target do
begin
if (i mod 4=0 )then
if (i mod 3=1)then
break;
write(i,',');
end;
writeln;
writeln('upon exit, i = ',i);
readln;
end.
```

خرج البرنامج السابق هو:

```
1,2,3,
upon exit, i = 4
```

الإجراء exit:

ينهي هذا الإجراء الكتلة block التي اسُتدعي فيها الإجراء exit ناقلاً التحكم إلى الكتلة التي استدعت هذه الكتلة التي أنهاها الإجراء exit فإذا كانت الكتلة هي البرنامج الرئيسي فإن استدعاء الإجراء exit ضمنه فإن ذلك سوف يؤدي إلى إنهاء تنفيذه.

و البرنامج التالي يوصح لنا عمل الإجراء exit:

Code
```
program test;
const target=20;
var i:integer;
f:text;const m='c:\f.txt';
procedure exittest;
begin
writeln;
for i:=1 to target do
begin
if(i=3)then
exit;
write(i,',');
end;
writeln('upon exit , i= ',i);
writeln('good bye');
end;
begin
exittest;
readln;
end.
```

خرج البرنامج السابق هو:

```
1,2,
```

الإجراء halt:

يقوم هذا الإجراء بإنهاء البرنامج مباشرة ناقلاً التحكم بذلك إلى نظام التشغيل operating system و يمكن استخدام هذا الإجراء لإعادة رقم إلى نظام التشغيل عند إنهاء البرنامج و هذا الرقم اختياري و يُمكن نظام التشغيل من معرفة الحالة التي انتهى عندها البرنامج. إذاً يمكن استدعاء الإجراء halt وفق صيغتين:

1. Halt;
2. Halt(9);

الإجراء الأول ينهي البرنامج عندما يصل إليه التنفيذ أما الآخر فينهي البرنامج و يعيد القيمة 9 إلى نظام التشغيل. و تعطي هذه الأرقام معاني يفرضها المبرمج و يفحصها نظام التشغيل عبر الأمر الذي يستخدم في الملفات الدفعية: if ERRORLEVEL

و المثال التالي يوضح لنا عمل الإجراء halt:

Code
```
program test;
var
```

```
c:char;
value:integer;
begin
repeat
write('enter the  value (1=halt)...');
readln(value);
if(value=1)then
halt;
until(false);
writeln(value);
writeln('good bye');
readln;
end.
```

سيتم الدخول في حلقة لا نهائية من أجل قيم صحيحة يدخلها المستخدم إلى أن يتم إدخال 1 فيتم إيقاف البرنامج مباشرة و نقل التحكم إلى نظام التشغيل.
ولن يتم طباعة أي شيْ على الشاشة.
أما بالنسبة لأوامر المقاطة المتعلقة بالتعليمة halt يمكننا البحث عن مرجع متخصص للتعرف على كيفية التعامل مع هذه الأوامر.

القفز غير المشروط –عبارة goto the unconditional jump –goto statement

يطلق على العبارة goto بعبارة القفز غير المشروط أو بعبارة التفرع غير المهيكل unstructured branch و ذلك لأنها تنقل التنفيذ من موقع إلى آخر مباشرة و بدون قيود أو شروط أو مراعاة لبنى أو تراكيب البرنامج.
تستخدم عبارة goto للقفز إلى اللافتة التي تعرف في قسم خاص ضمن قسم التصريح يدعى label.

Label <مجموعة من أسماء اللافتات>

لنأخذ اللافتات التالية مثالاً على تعريف اللافتات:

```
Label
Point1, jumtohere,A100,100;
```

بعد ذلك توضع اللافتة ضمن البرنامج في المكان الذي نريد تعلمه (تحديده) للقفز إليه فيما بعد تأتي عبارة goto لتستفيد من كون مكان القفز أصبح معرفاً فتطلب القفز إليه .

و البرنامج التالي يوضح لنا :

Code

```
program test;
var i:integer;
ch:char;
label retry, stop, next;
procedure getchar(message:string; var value:char);
begin
write(message,' ');
readln(value);
end;
begin
retry:
getchar('what is youe sex: <F,M>?... ',ch);
ch:=upcase(ch);
writeln(ch);
if not((ch='F')or(ch='M'))then
begin
writeln('enter the sex: <F,M>?... ');
goto retry;
```

```
end;
if(ch='M')then
writeln('sex is Male')
else
writeln('sex is Femal');
i:=i+1;
next:
i:=i+1;
if(i>20)then
goto stop;
write('.');
goto next;
stop:
writeln;
writeln;
writeln;
write('press enter ...');
readln;
end.
```

خرج البرنامج شبيه بالتالي:

```
what is youe sex: <F,M>?...  f
sex is Femal
...................

press enter ...
```

يسأل البرنامج السابق المستخدم عن جنسه فإذا كان ذكراً يكتب الحرف m أو M وذلك كاختصار للكلمة male أو الحرف f أو F وذلك كاختصار للكلمة female .
بعد ذلك يطبق البرنامج رسالة و عددا من النقاط (عدد هذه النقاط هنا هي 19 نفطة). أما إذا أدخلنا حرف غير الأحرف التي ذكرت فسيعيد البرنامج السؤال مرة أخرى.
لقد صرحنا عن ثلاث لافتات ضمن قسم التصريح label و استخدمناها ضمن البرنامج لتعليم أو لتحديد مواضع سوف يقفز إليها باستخدام عبارة goto فعندما يصل التنفيذ إلى عبارة goto يبحث عن المترجم عن موقع هذه اللافتة ضمن البرنامج و عند العثور عليها ينقل التحكم إلى التعليمة التالية للافتة.

لا تُستخدم عبارة goto كثيراً في البرامج بل و لا ينصح بها و يعتبرها البرمجون نقصاً في مقدارات المبرمج و لذلك يتجنبون استخدامها ضمن البرامج لما تسببه من إرباكات و تشعبات في البرنامج و تؤدي إلى صعوبة قي قراءته و فهمه.
ناهيك عن أن كل التفرعات التي تستخدم عبارة goto لبنائها يمكن بناؤها عبر عبارات لغة البرمجة المختلفة و بقوة أكبر.
و الآن سأقوم بتعديل البرنامج السابق بحيث يقوم بنفس العمل بعد حذف عبارات goto و استبدالها بما هو أفضل (حسب المبرمج):

Code

```
program test;
var i:integer;
ch:char;
```

```
procedure getchar(message:string; var value:char);
begin
write(message,' ');
readln(value);
end;
begin
repeat
getchar('what is youe sex: <F,M>?... ',ch);
ch:=upcase(ch);
if not((ch='F')or(ch='M'))then
writeln('enter the sex: <F,M>?... ');
until(ch='F')or(ch='M');
if(ch='M')then
writeln('sex is Male')
else
writeln('sex is Femal');
for i:=1 to 19 do
write('.');
writeln;
writeln;
writeln;
write('press enter ...');
readln;
end.
```

خرج البرنامج شبيه بالتالي:

```
what is youe sex: <F,M>?...  M
sex is Male
...................


press enter ...
```

اكتب برنامج يقوم بما يلي:
- إجرائية input_data لتعبئة مصفوفة من الارقام الحقيقية.
- إجرائية print لطباعة عناصر هذه المصفوفة على شاشة الخرج لقياسي و هي الشاشة.
- إجرائية small_value لحساب أصغر عنصر ضمن عناصر هذه المصفوفة.

Code

```
program test;
const n=10;
type
mat=array[1..n] of real;
function small_value(balance:mat;n:integer):real;
var small:real;i:integer;
begin
small:=balance[1];
for i:=2 to n do
if small>balance[i] then
small:=balance[i];
small_value:=small;
end;
procedure print(balance:mat; n:integer);
var i:integer;
begin
for i:=1 to n do
write(balance[i]:6:2,',');
writeln;
end;
procedure input_data(var balance:mat; n:integer);
var i:integer;
begin
for i:=1 to n do
begin
write('balance[',i,']= ');
readln(balance[i]);
end;
end;
var balance:mat;
begin
input_data(balance,10);
print(balance,10);
writeln;
writeln('searching');
writeln('the smallest value in the given array is = ',small_value(balance,10):6:2);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
balance[1]= 100.00
balance[2]=  40.00
balance[3]= -30.00
balance[4]= 400.00
balance[5]=  60.00
balance[6]= -25.00
balance[7]= -24.00
balance[8]=   0.00
balance[9]=   3.24
balance[10]=   0.50
100.00, 40.00,-30.00,400.00, 60.00,-25.00,-24.00,  0.00,  3.24,  0.50,

searching
the smallest value in the given array is = -30.00
```

لاحظ في البرنامج التالي أن التابعين فيه متكافئيين تماماً:

Code
program test; function sum_n(n:integer):integer; var i,res:integer; begin res:=0; i:=0; repeat res:=res+i; i:=i+1; until(i>=n+1); sum_n:=res; end; function sum_n1(n:integer):integer; var i,res:integer; begin res:=0; i:=0; while(i<=n)do begin res:=res+i; i:=i+1; end; sum_n1:=res; end; begin writeln(sum_n(100)); writeln(sum_n1(100)); readln; end.

خرج البرنامج السابق هو:

```
5050
5050
```

المهندس خالد ياسين الشيخ

ليكن لدينا البرنامج التالي:

Code
program test; var i,j:integer; begin for i:=1 to 5 do case i mod 4 of 1..2:begin write(i,','); break; end; 3..4:write(i,','); end; readln; end.

خرج البرنامج هو:

- **A. <u>1,</u>**
- B. 1,2,3,5,
- C. 1,2,3,
- D. 1,5,
- E. None of the above

اكتب إجراء يقوم بجمع عددين صحيحين موجبين مع توضيح عملية الاستدعاءات من أجل x=3 and y=4 :

Code
procedure sum(x,y:integer;var res:integer); begin if(x>0)then begin sum(x-1,y,res); res:=res+1 end else res:=y end;

توضيح كيفية الاستدعاءات: العودية تعتمد على مفهوم المكدس:

```
sum(3,4,res); ──► res=6+1=7
sum(2,4,res); ──► res=5+1=6
sum(1,4,res); ──► res=4+1=5
sum(0,4,res); ──► res=4  (*stop recursive توقف العودية*)
```

ما هو خرج البرنامج التالي:

اكتب تابع يقوم بجمع عددين صحيحين موجبين مع توضيح عملية الاستدعاءات من أجل x=3 and y=4 :

Code

```
function sum(x,y:integer):integer;
begin
if(x>0)then
sum:=1+sum(x-1,y)
else
sum:=y
end;
```

توضيح كيفية الاستدعاءات:

```
sum=1+sum(2,4);  → sum=1+6=7
sum=1+sum(1,4);  → sum=1+5=6
sum=1+sum(0,4);  → sum=1+4=5
  sum=4   (*توقف العودية*)
```

ما هو خرج البرنامج التالي مع توضيح كيفية الاستدعاءات:

Code

```
procedure proce(x,y:integer; VAR res:integer);
begin
if(y>0)then
begin
x:=x+1;
proce(x-1,y-1,res);
res:=res+x
end
else
res:=x+y;
end;
begin
proce(4,3,z);
writeln(z);
readln
end.
```

خرج البرنامج هو 19 .

توضيح كيفية الاستدعاءات:

```
      5
proce(4,3,res); → res=14+5=19
      5
proce(4,2,res); → res=9+5=14
      5
proce(4,1,res); → res=4+5=9

proce(4,0,res); → res=4+0=4
```

اكتب تابع يقوم بضرب عددين صحيين مع بيان كيفية الاستدعاء من أجل x=3 and y=4

Code

```
function mult(x,y:integer):integer;
begin
if(x>0)then
mult:=y+mult(x-1,y)
else
mult:=x*y
end;
```

بيان كيفية الاستدعاء:

```
mult=4+mult(2,4);  → mult=4+8=12

mult=4+mult(1,4);  → mult=4+4=8

mult=4+mult(0,4);  → mult=4+0=4

mult=0*4=0
```

اكتب برنامج يتضمن:

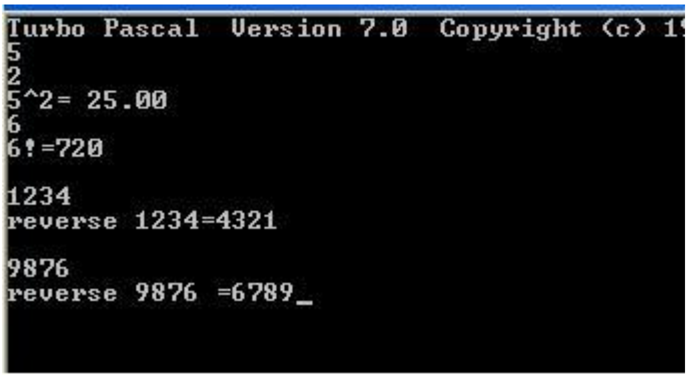
1. تابع power لحساب العلاقة x^n حيث x عدد كسري و n عدد صحيح موجب حيث $x^0=1$.
2. إجرائية fact لحساب العلاقة 1×2×3×.......(n-2)×(n-1)×n!=n حيث 0!=1.
3. تابع reverse لعكس أرقام عدد صحيح مدخل مثلاً 123 تصبح 321 دون استخدام بنية المصفوفات.
4. إجرائية reverse1 لعكس أرقام عدد صحيح مدخل باستخدام بنية المصفوفات(النسق).

Code

```
function power(x:real;n:integer):real;
begin
if n=0 then
power:=1
else
power:=x*power(x,n-1)
end;
procedure fact(n:integer; var res:integer);
begin
if n=0 then
res:=1
else
begin
fact(n-1,res);
res:=res*n;
end;
end;
function reverse(n:integer):integer;
var res:integer;
begin
res:=0;
while(n<>0)do
begin
res:=res*10+n mod 10;
n:=n div 10;
end;
reverse:=res;
```

```
end;
procedure reverse1(n:integer);
var info:array[1..8] of integer;
i,res:integer;
begin
i:=0;
while(n<>0)do
begin
i:=i+1;
info[i]:=n mod 10;
n:=n div 10;
end;
for res:=1 to i do
write(info[res]);
end;
writeln(x,'^',y,'=',power(x,y):6:2);
readln(x);
fact(x,z);
writeln(x,'!=',z);
readln(x);
writeln('reverse ',x,'=',reverse(x));
readln(x);
write('reverse ',x,' =');
reverse1(x);
readln
end.
```

صورة من خرج البرنامج هو:



ما هو خرج البرنامج التالي مع توضيح كيفية الاستدعاءات من أجل x=2 and y=5

Code

```
program test;
procedure proce(x,y:integer; var res:integer);
begin
if(x>0)then
begin
proce(x-1,y,res);
y:=y+1;
proce(x-1,y,res);
res:=res+y;
end
else
res:=y;
end;
var res:integer;
begin
proce(2,5,res);
writeln(res);
readlnulgdm
end.
```

خرج البرنامج هو 20

توضيح عملية الاستدعاءات:

العودية تعتمد على مفهوم المكدس:

```
proce(2,5,res);   (5 crossed out, 6 written above)
proce(1,5,res);
proce(0.5.res);
res=5

proce(0,6,res);
res=6
res=6+6=12

proce(1,6,res);
proce(0,7,res);
res=7

proce(0,7,res);
res=7
res=7+7=14
res=14+6=20
```

اكتب برنامج بلغة باسكال يتضمن تابع اسمه age_date يرد عمر الإنسان بالأيام باعتبار عدد أيام السنة 365 يوم و عدد أيام الشهر 30 يوم.

Code

```
program test;
function age_date(year,monthe,day:word):word;
begin
age_date:=year*365+monthe*30+day;
end;
var y,m,d:integer;
begin
writeln('enter your birthday');
write('enter your age years number(example 22)...');
readln(y);
write('enter the your age monthe...');
readln(m);
write('enter the your days...');
readln(d);
writeln(age_date(y,m,d));
readln
end.
```

اكتب برنامج يتضمن برمجة لعبة بسيطة على النحو التالي:
يقوم البرنامج بتوليد عدد عشوائي ثم يقوم المستخدم بإدخال عدد (تخمين) لمعرفة العدد الذي تم توليده فإذا كان العدد المدخل من قبل المستخدم أصغر من العدد المولد عشوائيا من قبل البرنامج يظهر البرنامج رسالة enter the number large from و إذا كان أكبر يظهر الرسالة enter the number small from وإلا يطبع البرنامج اسم المستخدم مع طباعة عدد محاولات المستخدم حتى عرف العدد المولد من قبل البرنامج.

Code

```
program test;
var
a,b,m:integer;
var s:string[30];
procedure game(n:integer);
var a,i:integer;
begin
i:=1;
writeln('enter the  number');
readln(a);
while(a<>n)do
begin
if(a<n)then
begin
writeln('enter the number large from',a:4);
readln(a);
end
else
begin
writeln('enter the number small from',a:4);
readln(a);
end;
i:=i+1;
end;
```

```
if i in [1..3] then
writeln(s,' very very good number of try = ',i:4)
else
writeln(s,' good number of try count is ',i:4);
end;
var n:integer;

begin
randomize;
n:=random(300);
writeln('enter your name');
readln(s);
{writeln(n); }
game(n);
readln
end.
```

صورة من تنفيذ البرنامج



اكتب إجراء عودي يقوم بضرب عددين صحيحين موجبين.

Code

```
program test;
var res:integer;
procedure mult(x,y:integer);
begin
if(x>0)then
begin
mult(x-1,y);
res:=res+y
end
else
res:=x
end;
var a,b:integer;
begin
readln(a,b);
```

```
mult(a,b);
writeln(res);
readln
end.
```

توضيح الاستدعاءات من أجل a=3 and b=6 :

```
mult(3,6); → res=12+6=18
mult(2,6); → res=6+6=12
mult(1,6); → res=0+6=6
mult(0,6); (*توقف العودية*)
res=0
```

وبالتالي قيمة المتحول res تكون 18.

اكتب إجرائية عودية total تقوم بحساب مجموع الأعداد من n..1 حيث n عدد صحيح موجب.

Code

```
program test;
procedure total(n:integer; var s:integer);
begin
if(n>0)then
begin
total(n-1,s);
s:=s+n;
end
else
s:=0
end;
var n,res:integer;
begin
readln(n);
total(n,res);
writeln(res);
readln
end.
```

توضيح الاستدعاءات من أجل n=4

```
total(4,s) → s=6+4=10
total(3,s) → s=3+3=6
total(2,s) → s=1+2=3
total(1,s) → s=0+1=1
total(0,s) (*توقف العودية*)
s=0
```

وبالتالي res=s=10

اكتب إجرائية عودية dec-bin لتحويل عدد عشري صحيح إلى ثنائي و إجرائية عودية bin-dec لتحويل عدد ثنائي إلى عشري دون استخدام أي بنية معطيات باستثناء المتحولات العادية.

Code

```
program test;
procedure dec_bin(n:integer);
begin
if(n<>0)then
begin
dec_bin(n div 2);
write(n mod 2);
end
end;
function pow(x,n:integer):longint; forward;
function bin_dec(n:integer;i:integer):longint;
begin
if(n=0)then
bin_dec:=0
else
bin_dec:=n mod 10*pow(2,i)+bin_dec(n div 10,i+1);
end;
function pow(x,n:integer):longint;
var res,i:longint;
begin
res:=1;
for i:=1 to n do
res:=res*x;
pow:=res
end;
var n:integer;
begin
readln(n);
dec_bin(n);
writeln;
writeln('enter the number binary 0..1 only example 1001');
readln(n);

writeln(bin_dec(n,0));
readln
end.
```

ما هو خرج البرنامج التالي:

Code

```
program test;
procedure sum(n:integer; var res:integer);
begin
if(n=0)then
res:=0
else
begin
sum(n div 2,res);
res:=res+n mod 10;
end;
end;
var e:integer;
begin
e:=10;
sum(123,e);
writeln(e);
readln
end.
```

خرج البرنامج هو:

a. **20**
b. 30
c. 6
d. 18
e. None of the above

بيان كيفية عملية الاستدعاءات :

```
sum(123,res) →  res=17+3=20
sum(61,res)  →  res=16+1=17
sum(30,res)  →  res=16+0=16
sum(15,res); →  res=11+5=16
sum(7,res);  →  res=4+7=11
sum(3,res);  →  res=1+3=4
sum(1,res);  →  res=0+1=1
sum(0,res);     (*stop recursive*)

res=0
```

و بالتالي res=e=20

ما هو خرج البرنامج التالي مع بيان كيفية عمليات الاستدعاء للإجرائية:

Code

```
program test;
procedure proc(x,y:integer;var res:integer);
begin
if(x>0)then
begin
y:=y+1;
proc(x-1,y+2,res);
y:=y+1;
res:=res+x+y;
end
else
res:=y
```

```
end;
var res:integer;
begin
proc(2,3,res);
writeln(res);
readln;
end.
```

بيان كيفية الاستدعاء:

```
proc(2,5,res);  →  res=18+2+5=25
       ↑
proc(1,8,res);  →  res=9+1+8=18

proc(0,9,res);
     ↙
   res=9
```

وبالتالي res=25 و يكون خرج البرنامج هو 25

ما هو خرج البرنامج و ما هو عمل الإجرائية xxxx :

Code

```
program test;
procedure xxxx(x,y:integer;var res:integer);
begin
if(y=0)then
res:=x
else
begin
xxxx(x,y-1,res);
res:=x+y;
end;
end;
var res:integer;
begin
xxxx(7,2,res);
writeln(res);
readln
end.
```

توضيح الاستدعاءات:

```
xxxx(7,2,res);  →  res=7+2=9
       ↑
xxxx(7,1,res);  →  res=7+1=8
xxxx(7,0,res);  (*توقف العودية*)
   ↙
res=7+0=7
```

و بالتالي res=9
خرج البرنامج هو 9 و عمل الإجرائية هو جمع عددين صحيحين.

برنامج ضرب و قسمة و جمع و جمع و طرح عددين حقيقيين:

Code

```
program test;
var c:char;
n:integer;
a,b:real;
function sum(x,y:real):real;
begin
sum:=x+y;
end;
function sub(x,y:real):real;
begin
sub:=x-y;
end;
function mult(x,y:real):real;
begin
mult:=x*y;
end ;
function division(x,y:real):real;
begin
division:=x/y;
end;
begin

repeat
writeln(' 1 for sum , 2 for subtract , 3 for mult , 4 for division');
readln(n);
while( not (n in [1..4]))do
begin
writeln('enter 1..4');
readln(n);
end;
case n of
1:begin
writeln('enter tow number for sum');
readln(a,b);
writeln(a:6:2,'+':3,b:6:2,'=':3,sum(a,b):6:2);
end;
2:begin
writeln('enter tow number for sub');
readln(a,b);
writeln(a:6:2,'-(':3,b:6:2,')=':3,sub(a,b):6:2);
end;

3:begin
writeln('enter tow number for mult');
readln(a,b);
writeln(a:6:2,'*':3,b:6:2,'=':3,mult(a,b):6:2);
end;
4:
begin
writeln('enter tow number for division');
readln(a,b);
```

```
if(b=0)then
writeln('error')
else
begin
writeln(a:6:2,'/':3,b:6:2,'=':3,division(a,b):6:2);
end;
end;
end;
writeln('do you want exit press y ');
readln(c);

c:=upcase(c);
until(c='Y')or (c='y');
end.
```

اكتب إجرائية print_prime تقوم بطباعة الأعداد الأولية ضمن المجال [1..n] حيث n عدد صحيح موجب أكبر تماما من 1.
ملاحظة: العدد الأولي لا يقبل القسمة إلا على نفسه و على الواحد.

Code

```
program test;
function check_prime(n:integer):boolean;
var i:integer;
begin
check_prime:=true;
for i:=2 to n div 2 do
if n mod i=0 then
begin
check_prime:=false;
break;
end
else
check_prime:=true;
end;
procedure print_prime(n:integer);
var i:integer;
begin
i:=2;
while(i<=n)do
begin
if i mod 25=0then
writeln;
if(check_prime(i))then
write(i,',');
i:=i+1;
end;
end;
var n:integer;
begin
n:=500;
print_prime(n);
readln
```

```
end.
```

خرج البرنامج السابق هو:

```
2,3,5,7,11,13,17,19,23,
29,31,37,41,43,47,
53,59,61,67,71,73,
79,83,89,97,
101,103,107,109,113,
127,131,137,139,149,
151,157,163,167,173,
179,181,191,193,197,199,
211,223,
227,229,233,239,241,
251,257,263,269,271,
277,281,283,293,
307,311,313,317,
331,337,347,349,
353,359,367,373,
379,383,389,397,
401,409,419,421,
431,433,439,443,449,
457,461,463,467,
479,487,491,499,
```


اكتب إجرائية تقوم بطباعة قواسم الأعداد ضمن المجال [1..n] حيث ى عدد صحيح موجب أكبر تماما من 1:

Code

```
program test;
procedure divi(n:integer);
var i:integer;
begin
for i:=1 to n div 2 do
if n mod i=0 then
write(i,',');
end;
var i,n:integer;
begin
n:=25;
i:=2;
for i:=2 to n do
begin
write(i,'=':3);divi(i);
writeln;
```

```
end;
readln;
end.
```

خرج البرنامج السابق هو:

```
2   =1,
3   =1,
4   =1,2,
5   =1,
6   =1,2,3,
7   =1,
8   =1,2,4,
9   =1,3,
10  =1,2,5,
11  =1,
12  =1,2,3,4,6,
13  =1,
14  =1,2,7,
15  =1,3,5,
16  =1,2,4,8,
17  =1,
18  =1,2,3,6,9,
19  =1,
20  =1,2,4,5,10,
21  =1,3,7,
22  =1,2,11,
23  =1,
24  =1,2,3,4,6,8,12,
25  =1.5.
26  =1,2,13,
27  =1,3,9,
28  =1,2,4,7,14,
29  =1,
30  =1,2,3,5,6,10,15,
31  =1,
32  =1,2,4,8,16,
33  =1,3,11,
34  =1,2,17,
35  =1,5,7,
36  =1,2,3,4,6,9,12,18,
37  =1,
38  =1,2,19,
39  =1,3,13,
40  =1,2,4,5,8,10,20,
41  =1,
42  =1,2,3,6,7,14,21,
43  =1,
44  =1,2,4,11,22,
45  =1,3,5,9,15,
46  =1,2,23,
47  =1,
48  =1,2,3,4,6,8,12,16,24,
49  =1,7,
50  =1,2,5,10,25,
```

اكتب برنامج يقوم بطباعة رمز النجمة * على الشكل أدناه من أجل n=9 مثلاً:

```
*********
********
*******
******
*****
****
***
**
*
```

Code

```
program test;
procedure print(n:integer);
var i,j:integer;
begin
for i:=n downto 1 do
begin
for j:=1 to i do
write('*');
writeln;
end;
end;
begin
print(20);
readln;
end.
```

اكتب برنامج يقوم بطباعة رمز النجمة * على الشكل أدناه من أجل n=9 مثلاً:

```
*
**
***
****
*****
******
*******
********
*********
```

Code

```
program test;
var f:text;
procedure print(n:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
for j:=1 to i do
write(f,'*');
writeln(f);
end;
end;
```

```
begin
assign(f,'c:\f.txt');
rewrite(f);
print(9);
close(f);
readln;
end.
```

حساب و تدوير إلى أعداد من مضاعفات الـ 2:

Code

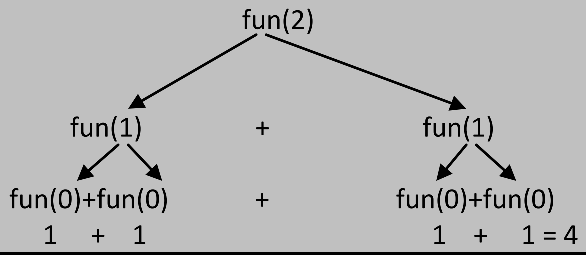
```
program test;
function rounding(n:integer):integer;
begin
if n mod 2=0 then
rounding:=n
else
rounding:=n+(2-n mod 2);
end;
var
n:integer;
begin
readln(n);
writeln(rounding(n));
readln
end.
```

وضح كيفية الاستدعاءات في التابع التالي:

Code

```
program test;
function fun(n:integer):integer;
begin
if(n=0)then
fun:=1
else
fun:=fun(n-1)+fun(n-1);
end;
begin
writeln(fun(2));
readln
end.
```

توضيح الاستدعاءات:



أكتب إجرائية power لحساب العلاقة x^n مثال:

5^2=25 5^{-2}=0.04

Code

```
procedure pow(x,n:integer;var y:real);
var c,i:real;
begin
c:=1;
i:=0;
```

$$\sum_{i=0}^{n-1} 1 = n = O(n)$$

```
if(n>=0)then
begin
while(i<n)do
begin
c:=c*x;
i:=i+1;
end;
y:=c;
end
else
begin
while(i<>n)do
```

$$\sum_{i=n<0}^{0} 1 = n = O(n)$$

```
begin
c:=c*1/x;
i:=i-1;
end;
y:=c;
end;
end;
var r:real;
x,n:integer;
begin
readln(x,n);
pow(x,n,r);
writeln(r:8:4);
readln
end.
```

ليكن لدينا التابعان التاليين:

Code

```
program test;
type
matrix=array[1..50,1..50] of integer;
function f1(x,n:integer):integer;
var res:integer;
begin
res:=1;
while(n>0)do
begin
res:=res*x;
n:=n-1;
end;
f1:=res
end;
function xxxx(mat:matrix;n:integer):integer;
var i,j,res:integer;
begin
res:=0;
i:=1;
while(i<=n)do
begin
for j:=1 to i-1 do
res:=res+mat[i,j];
i:=i+1;
end;
xxxx:=res
end;
```

و المطلوب:

1. ما هو عمل البرنامج f1 و ما هي درجة تعقيده.
2. اكتب تابع عودي (تراجعي) من التابع f1.
3. ما هو عمل التابع xxxx و ما هي درجة تعقيده.

الحل:

من أجل x=5 and n=3 يكون:

أول دخول ضمن الحلقة:

res=1*5=5
n=2

ثاني دخول ضمن الحلقة:

res=5*5=25
n=1

ثالث و آخر دخول ضمن الحلقة:

res=25*5=125
n=0

ومنه:

f1=res=125

و بالتالي عمل التابع f1 هو رفع إلى قوة x^n

نقوم بحساب تعقيد التابع كالتالي:

رقم الدخول	قيمة n
1	n
2	n-1
3	n-2
⋮	⋮
J+1	n-j

آخر مرة يدخل فيها الحلقة: n-j=1

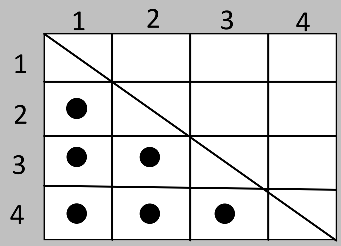
ومنه : عدد العمليات هو j+1=n ≈ O(n)

نكتب تابع عودي :

```
function f2(x,n:integer):integer;
var res:integer;
begin
if(n=0)then
f2:=1
else
f2:=x*f2(x,n-1);
end;
```

أما عمل التابع xxxx يقوم بحساب مجموع عناصر ما تحت القطر الرئيسي.

مثلا من أجل مصفوفة مربعة 4×4=16 elements



$$\sum_{i=1}^{n}\sum_{j=1}^{i-1}1=\sum_{i=1}^{n}i-1=\sum_{i=1}^{n}i-\sum_{i=1}^{n}1=\frac{n(n+1)}{2}-n=\frac{n^2+n-2n}{2}$$

$$=\frac{n^2-n}{2}\cong O(n^2)$$

اكتب إجرائية عودية تقوم بضرب عددين اعتماداً على طريقة التنصيف:
r=x*y
1. y زوجي: r=x*2* y div 2;
2. y فردي: r=(x*2)*(y div 2)+x

Code

```
program test;
function mult(x,y:integer):integer;
begin
if(y=0)then
mult:=0
else
if(y mod 2=0)then
mult:=mult(x*2,y div 2)
else
mult:=x+mult(x*2,y div 2);
end;
```

توضيح عملية الاستدعاء من أجل x=3 and y=4:

```
mult(3,4)
mult:=mult(6,2)
mult:=mult(12,1);
mult:=12+mult(24,0);=12+0=12
          ↙
     mult:=0;
procedure mult1(x,y:integer; var res:integer);
begin
if(y=0)then
res:=0
else
if(y mod 2=0)then
mult1(x*2,y div 2,res)
else
begin
mult1(x*2,y div 2,res);
res:=res+x;
end;
end;
```

توضيح عملية الاستدعاء من أجل x=3 and y=4:

```
mult1(3,4,res)
mult1(6,2,res)
mult(12,1,res) ⟶ res=0+12=12
mult(24,0,res);
   ↙
 res=0
function mult2(x,y:integer):integer;
var res,t:integer;
begin
t:=x;
res:=0;
while(y<>0)do
begin
```

```
if (y mod 2=1)then res:=res+t;
t:=t*2; y:=y div 2;
end;
mult2:=res;
end;
```

أكتب تابع عودي و تابع تكراري لإيجاد أصغر عنصر في نسق من الأعداد الصحيحة:

مثال من أجل نسق يحوي خمس عناصر كالتالي:

	1	2	3	4	5
mat	99	77	55	66	70

على التابعين أن يعيدا أصغر قيمة و هي هنا 55 .

Code

```
program test;
type
matrix=array[1..50] of integer;
var a:matrix; n,i:integer;
function min_value1(mat:matrix;n:integer):integer;
begin
if(n=1)then
min_value1:=mat[1]
else
begin
min_value1:=min_value1(mat,n-1);
for i:=1 to n-1 do
if(mat[i+1]<mat[i])then
min_value1:=mat[i+1];
end;
end;
function min(mat:matrix; n:integer):integer;
var i,j,k:integer;
begin
i:=1;
j:=i;
for k:=i+1 to n do
if(mat[k]<mat[j])then
j:=k;
min:=mat[j];
end;
```

اكتب إجرائية رفع إلى قوة (تخفيض التعقيد):

Code

```
program test;
function pow(x,n:integer):integer;
begin
if(n=0)then
pow:=1
else
if(n=1)then
pow:=x
else
if(n mod 2=0)then
pow:=pow(x*x,n div 2)
else
pow:=pow(x*x,n div 2)*x;
end;
```

توضيح عملية الاستدعاء من أجل x=3 and y=4:

```
pow(3,4);
pow:=pow(9,2);
pow:=pow(81,1);
  ↙
  pow:=81
```

نحولها إلى إجرائية:

```
procedure pow1(x,n:integer; var res:integer);
begin
if(n=0)then
res:=1
else
if(n mod 2=0)then
pow1(x*x,n div 2,res)
else
begin
pow1(x*x,n div 2,res);
res:=res*x
end;
end;
```

توضيح عملية الاستدعاء من أجل x=3 and y=4:

```
Pow1(3,4,res);

Pow1(9,2,res);

Pow1(81,1,res); ⟶ res=1*81=81

Pow1(6561,0,res); (*توقف العودية*)
  ↙
    Res=1
```

نكتب تابع تكراري من الإجرائية السابقة:

```
function pow2(x,n:integer):integer;
```

```
var t,res:integer;
begin
t:=x; res:=1;
while(n<>0)do
begin
if(n mod 2=1)then res:=res*t;
t:=t*t; n:=n div 2;
end;
pow2:=res;
end;
```

أكتب إجرائية عودية لإيجاد أكبر عدد و أصغر عدد و المتوسط الحسابي لمجموعة من الأعداد :

Code

```
program test;
var
n,i:integer;
xmax,xmin,xmean,x:real;
procedure account_values(x:real; n,i:integer; var max,min,mean:real);
begin
if(i<n)then
begin
write('enter the value...',i+1,':');
readln(x);
if(min>x)then
min:=x;
if(max<x)then
max:=x;
mean:=mean+x;
account_values(x,n,i+1,max,min,mean);
end;
end;
begin
writeln('enter the count of the number for input');
readln(n);
i:=1;
if(i<=n)then
begin
write('enter the value...',i,':');
readln(x);
xmax:=x;xmin:=x;xmean:=x;
account_values(x,n,i,xmax,xmin,xmean);
writeln('max   =':5,xmax:6:2);
writeln('min   =':5,xmin:6:2);
writeln('xmean =':5,xmean/n:6:2);
end
else
writeln('error');
readln
```

```
end.
```

صورة نتائج تنفيذ البرنامج السابق:

```
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
enter the count of the number for input
3
enter the value...1:16
enter the value...2:19
enter the value...3:17
max    = 19.00
min    = 16.00
xmean = 17.33
```

```
enter the count of the number for input
4
enter the value...1:1.6
enter the value...2:1.4
enter the value...3:16
enter the value...4:18
max    = 18.00
min    =  1.40
xmean =  9.25
```


اكتب إجرائية عودية تقوم بطباعة عناصر مصفوفة شعاع بشكل معكوس:

اكتب إجرائية عودية لحساب القاسم المشترك الأعظم لعددين صحيحين موجبين :

Code

```
program test;
type
matrix=array[1..200] of integer;
procedure print_array(mat:matrix;index,n:integer);
begin
if(index<=n)then
begin
{write(mat[index],',');}
print_array(mat,index+1,n);
write(mat[index],',');(* print reverse*)
end;
end;
begin
print_array(mat,1,5);
end.
function gcd(m,n:integer):integer;
begin
if(m=n)then
gcd:=m
else
begin
if(m>n)then
m:=m-n
else
n:=n-m;
gcd:=gcd(m,n);
end;
end;
begin
```

```
writeln(gcd(90,49));
readln;
end.
```

اكتب إجرائية فرز بالفقاعات (التعويم) Bubble sort لنسق.
اكتب إجرائية movable تقم بتحويل مصفوفة مربعة إلى منقول مصفوفة.
مثال: مصفوفة 2×2

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \Longrightarrow \begin{bmatrix}1 & 3\\2 & 4\end{bmatrix}$$

اكتب تابع series تقوم بحساب الدالة التالية:

$$\sum_{i=1}^{n}\frac{1}{i}$$

Code

```
program test;
const n=15;
type
matrix=array[1..n,1..n]of integer;
mat=array[1..n] of integer;
procedure movable(a:matrix;var b:matrix; n:integer);
var i,j:integer;
begin
for i:=1 to n do
for j:=1 to n do
b[i,j]:=a[j][i]
end;
procedure Bubble_sort(var a:mat;n:integer);
var temp,i:integer; flag:boolean;
begin
repeat
flag:=true;
for i:=1 to n-1 do
if a[i]>a[i+1] then
begin
temp:=a[i];
a[i]:=a[i+1];
a[i+1]:=temp;
flag:=false;
end;
until(flag);
end;
function series(n:integer):real;
var sum:real; i:integer;
begin
sum:=0;
for i:=1 to n do
sum:=sum+1/i;
series:=sum;
end;
```

لتكن لدينا المصفوفة M[1..n][1..n] من الأعداد الطبيعية تحتوي على n^2 عنصر بلا أي ترتيب خاص نريد تحويل هذه المصفوفة إلى مصفوفة مفروزة وفق الفرز التالي:

يجب أن كون العناصر مرتبة في كل سطر :

M[x][y]<M[x][y+1] for all x and y

يجب أن تكون العناصر مرتبة في كل عمود:

M[x][y]<M[x+1][y] for all x and y

اكتب إجرائية فرز المصفوفة وفق الترتيب السابق و ما هو تعقيد الإجرائية ؟؟

Code

```
program test;
const n=15;
type
matrix=array[1..n,1..n]of integer;
procedure sort_row(var m:matrix; n:integer);
var i,j,k,temp:integer;
begin
for k:=1 to n do
for i:=1 to n do
for j:=1 to n-1 do
if(m[i,j]>m[i,j+1])then
begin
temp:=m[i,j];
m[i,j]:=m[i,j+1];
m[i,j+1]:=temp;
end;
end;
procedure sort_column(var m:matrix; n:integer);
var i,j,k,temp:integer;
begin
for k:=1 to n do
for i:=1 to n-1 do
for j:=1 to n do
if(m[i,j]>m[i+1,j])then
begin
temp:=m[i,j];
m[i,j]:=m[i+1,j];
m[i+1,j]:=temp;
end;
end;
procedure print(m:matrix;n:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
for j:=1 to n do
write(m[i,j]:4);
writeln;
end;
end;
var i,j:integer;
m:matrix;
begin
```

```
for i:=1 to 3 do
for j:=1 to 3 do
readln(m[i][j]);
print(m,3);
sort_row(m,3);
writeln;
print(m,3);
sort_column(m,3);
writeln;
print(m,3);
readln
end.
```

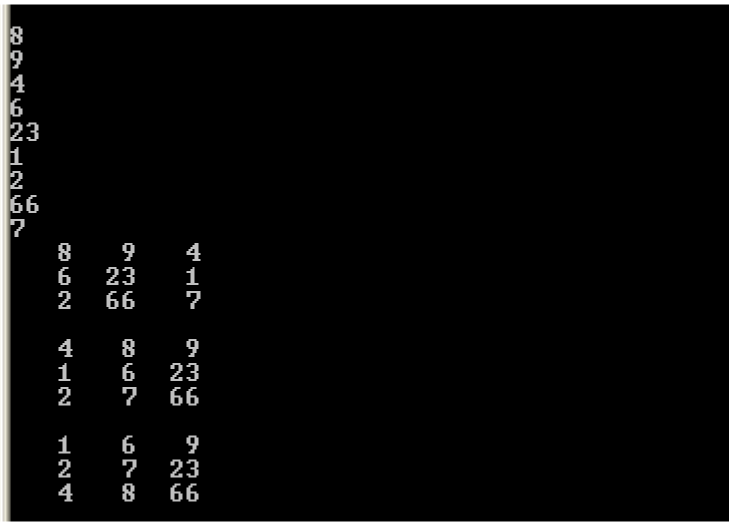
حساب عدد عمليات المقارنة بأسوأ الأحوال و مرتبة التعقيد:

$$\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n-1}1=\sum_{k=1}^{n}\sum_{i=1}^{n}n-1=\sum_{k=1}^{n}n(n-1)=$$

$$n^2(n-1)=n^3-n^2\cong O(n^3)$$

صورة من نتيجة تنفيذ البرنامج بفرض لدينا مصفوفة مربعة 3×3=9 elements عناصرها كالتالي:

$$\begin{Bmatrix} 8 & 9 & 4 \\ 6 & 23 & 1 \\ 2 & 66 & 7 \end{Bmatrix}$$



اكتب برنامج بلغة تربو باسكال لإيجاد القاسم المشترك الأعظم لـ N عدد صحيح موجب مدخل لا يساوي الصفر.
مثال: القاسم المشترك الأعظم للأعداد 90 , 120 , 60 , 150 , 180 هو 30.
القاسم المشترك الأعظم للعددين 20 , 5 هو 5.

Code

```
program test;
const max=40;
function GCD(a,b:word):integer;
begin
while(a<>b)do
begin
if(a>b)then
a:=a-b
else
b:=b-a;
end;
GCD:=a;
end;
function GCD_N(n:word):integer;
var i:integer; f,a:array[1..max]of integer;
begin
for i:=1 to n do
begin
write('enter the number',i,' ');
readln(a[i]);
end;
f[1]:=gcd(a[1],a[2]);
for i:=1 to n do
f[i+1]:=gcd(a[i],f[i]);
GCD_N:=f[n+1];
end;
var i,n:integer;
begin
writeln('enter the count number>=2');
readln(n);
while(2>n)do
begin
writeln('enter the number>=2');
readln(n);
end;
writeln(GCD_N(n));
readln;
end.
```

صورة من تنفيذ البرنامج السابق:

```
enter the count number>=2
2
enter the number1
20
enter the number2
5
GCD(20,5)=5


enter the count number>=2
3
enter the number1
54
enter the number2
66
enter the number3
48
GCD(54,66,48)=6

enter the count number>=2
5
enter the number1
90
enter the number2
120
enter the number3
60
enter the number4
150
enter the number5
180
GCD(90,120,60,150,180)=30


enter the count number>=2
4
enter the number1
150
enter the number2
48
enter the number3
96
enter the number4
160
GCD(150,48,96,160)=2
```

اكتب برنامج بلغة تربو باسكال لإيجاد المضاعف المشترك الأصغر لـ N عدد صحيح موجب مدخل لا يساوي الصفر.
مثال: المضاعف المشترك الأصغر للأعداد 12 , 6 , 18 هو 36.
المضاعف المشترك الأصغر للأعداد 48 , 54 , 66 هو 4752.
المضاعف المشترك الأصغر للأعداد 8 , 2 هو 8.

Code

```
program test;
const max=40;
const m='c:\f.txt';
var t:text;
 f,a:array[1..max]of integer;
function GCD(a,b:word):integer;
begin
while(a<>b)do
begin
if(a>b)then
a:=a-b
else
b:=b-a;
end;
GCD:=a;
end;
function LCM(a,b:word):integer;
begin
LCM:=(a*b) div GCD(a,b);
end;
function LCM_N(n:word):integer;
var i:integer;
begin
for i:=1 to n do
begin
writeln(t,'enter the number',i,' ');
readln(a[i]);
writeln(t,a[i]);
end;
f[1]:=LCM(a[1],a[2]);
for i:=1 to n do
f[i+1]:=LCM(a[i],f[i]);
LCM_N:=f[n+1];
end;
var i,n,y:integer;
begin
assign(t,m);
rewrite(t);
writeln(t,'enter the count number>=2');
readln(n);
writeln(t,n);
```

```
while(2>n)do
begin
writeln(t,'enter the number>=2');
readln(n);
end;
y:=LCM_N(n);
for i:=1 to n do
begin
if(i=1)then
write(t,'LCM(',a[i],',')
else
if(i<>n)then
write(t,a[i],',')
else
write(t,a[i],')=');
end;
writeln(t,y);
close(t);
readln;
end.
```

صورة عن تنفيذ البرنامج السابق:

```
enter the count number>=2
3
enter the number1
48
enter the number2
54
enter the number3
66
LCM(48,54,66)=4752
```



```
enter the count number>=2
2
enter the number1
8
enter the number2
2
LCM(8,2)=8
```

```
enter the count number>=2
3
enter the number1
12
enter the number2
6
enter the number3
18
LCM(12,6,18)=36
```

ليكن لدينا التابع التالي:

Code

```
function strange(x,n:integer):integer;
var i,j,px,sum:integer;
begin
i:=2;
sum:=x;
while(i<=n)do
begin
j:=2;  px:=x;
while(j<=i)do
begin
px:=px*x; j:=j+1;
end;
sum:=sum+px; i:=i+1;
end;
strange:=sum;
end;
```

1. ما الذي يحسبه التابع.
2. ما هي درجة تعقيده.
3. أعد كتابة هذا التابع بحيث يحسب القيمة نفسها و لكن بعدد عمليات أقل.
4. ما هي درجة تعقيد التابع الجديد.

التابع يقوم بحساب العلاقة التالية:

$$\sum_{i=1}^{n} x^i = x^1 + x^2 + x^3 + \cdots \ldots\ldots\ldots\ldots + x^n \ : n > 0 ; n, x : integer;$$

ملاحظة: لدينا:

$$\sum_{i=0}^{n} x^i = \frac{(x^{n+1} - 1)}{(x-1)} : x \neq 1 \ and \ n \geq 0$$

نقوم بحساب عدد عمليات التابع (كلفة هنا هي عملية الضرب) :

$$\sum_{i=2}^{n}\sum_{j=2}^{i} 1 = \sum_{i=2}^{n} i - 1 = \sum_{i=2}^{n} i - \sum_{i=2}^{n} 1 = \frac{\big(n(n+1)\big)}{2} - 1 - (n-1)$$

$$= \frac{n^2+n-2n}{2} = \frac{n^2-n}{2} \cong O(n^2)$$

نقوم بكتابة تابع يحسب القيمة نفسها بعدد عمليات أقل:

```
function strange1(x,n:integer):integer;
var i,px,sum:integer;
begin
i:=1; px:=1;
sum:=0;
while(i<=n)do
begin
px:=px*x;
sum:=sum+px;
i:=i+1;
end;
```

```
strange1:=sum;
end;
```

نقوم بحساب عدد عمليات التابع (كلفة هنا هي عملية الضرب) :

$$\sum_{i=1}^{n} 1 = n \cong O(n)$$

احسب عدد عمليات و تعقيد كلا مما يلي:

```
For i ⟵ 1 to n do
  x ⟵ x+1;
```

$$\sum_{1=\le i\le=n} 1 = n \cong O(n)$$

```
For i ⟵ 1 to n do
  For j ⟵ 1 to n do
    x ⟵ x+1
```

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2 \cong O(n^2)$$

ليكن لدينا الخوارزميات التالية:

Code

الخوارزمية الأولى:

```
type
mat1=array[1..50,1..50]of integer;
var mat:mat1;
x1,n,i,j:integer;
readln(n);
x1:=0;
i:=1;
while(i<=n)do
begin
j:=1;
while(j<i)do
begin
x1:=x1+mat[i][j];
j:=j+1;
end;
i:=i+1;
end;
writeln((x1));
```

الخوارزمية الثانية:

```
type
mat2=array[1..50,1..50]of integer;
var mat:mat2;
x2,n,i,j:integer;
readln(n);
x2:=0;
i:=1;
while(i<=n)do
begin
j:=1;
while(j<=3)do
begin
x2:=x2+mat[i][j];
j:=j+1;
end;
i:=i+1;
end;
writeln((x2));
```

الخوارزمية الثالثة:

```
type
mat3=array[1..50,1..50]of integer;
var mat:mat3;
x3,n,i,j:integer;
readln(n);
x3:=0;
j:=n;
while(j>=1)do
begin
x3:=x3+mat[n-j+1][j];
j:=j-1;
end;
writeln((x3));
readln
end.
```

الخوارزمية الرابعة:

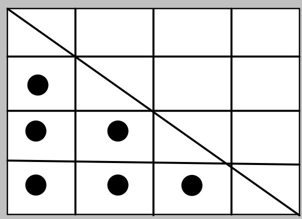
```
program test;
type
mat4=array[1..50,1..50]of integer;
var mat:mat4;
x4,n,i,j:integer;
readln(n);
x4:=0;
i:=1;
while(i<=n)do
begin
j:=n;
while(j>i)do
begin
x4:=x4+mat[i][j];
```

```
j:=j-1;
end;
i:=i+1;
end;
writeln((x4));
readln
end.
```

و المطلوب :

1 - ما هو عمل كل خوارزمية من الخوارزميات السابقة.
2 - ما هي عدد العمليات الجمع في كل خوارزمية و ما هي درجة التعقيد.

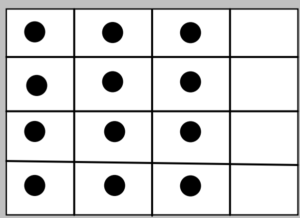
الخوارزمية الأولى: من أجل n=4



عمل الخوارزمية الأولى هو حساب مجموع عناصر ما تحت القطر الرئيسي لمصفوفة مربعة (المثلث السفلي)

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1}1=\sum_{i=1}^{n}i-1=\sum_{i=1}^{n}i-\sum_{i=1}^{n}1=\frac{n(n+1)}{2}-n=\frac{n^2+n-2n}{2}=\frac{n^2-n}{2}\cong O(n^2)$$

الخوارزمية الثانية: من أجل n=4

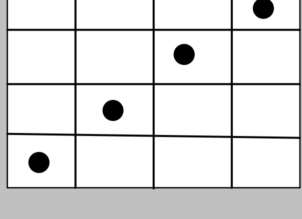


عمل الخوارزمية الثانية هو حساب مجموع عناصر الأعمدة الثلاثة الأولى فقط من المصفوفة المربعة .

$$\sum_{i=1}^{n}\sum_{j=1}^{3}1=\sum_{i=1}^{n}3=3n\cong O\,(n)$$

عدد العمليات

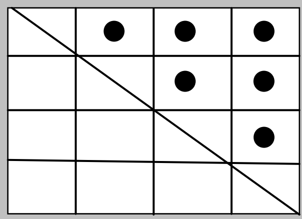
الخوارزمية الثالثة: من أجل n=4



عمل الخوارزمية الثالثة هو حساب مجموع عناصر القطر الثانوي في مصفوفة مربعة.

$$\sum_{j=1}^{n}1=n=O(n)$$

الخوارزمية الرابعة: من أجل n=4



عمل الخوارزمية الرابعة هو حساب مجموع عناصر ما فوق القطر الرئيسي لمصفوفة مربعة (المثلث العلوي).

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}1=\sum_{i=1}^{n}n-i=\sum_{i=1}^{n}n-\sum_{i=1}^{n}i=n^2-(\frac{n(n+1)}{2})=\frac{2n^2-n^2-n}{2}=\frac{n^2-n}{2}=O(n^2)$$

ملاحظة هامة:

$$\sum_{j=1}^{n}1=\sum_{j=1}^{i}1+\sum_{j=i+1}^{n}1$$

ومنه:

$$\sum_{j=i+1}^{n}1=\sum_{j=1}^{n}1-\sum_{j=1}^{i}1=n-i$$

ما هو عمل كلا من الخوارزميات التالية:
الخوارزمية الخامسة:

```
type
mat5=array[1..50,1..50]of integer;
var mat:mat5;
x5,n,i,j:integer;
readln(n);
for i:=1 to n do
x5:=x5+mat[i,i];
writeln(x5);
```

عمل هذه الخوارزمية حساب مجموع عناصر القطر الرئيسي لمصفوفة مربعة.
الخوارزمية السادسة:

```
type
mat6=array[1..50,1..50]of integer;
var mat:mat6;
x6,n,i,j:integer;
readln(n);
x6:=0;
for i:=n downto 1 do
x6:=x6+mat[i,n-i+1];
writeln(x6);
```

عمل هذه الخوارزمية حساب مجموع عناصر القطر الثانوي لمصفوفة مربعة.

احسب عدد عمليات في كلا مما يلي ودرجة التعقيد في أسوأ الأحوال:

Code

```
1.sum:=0;
For k:=1 to n do
  For j:=1 to sqr(n) do
     Sum++;
```

$$\sum_{k=1}^{n}\sum_{j=1}^{n^2} 1 = \sum_{k=1}^{n} n^2 = n^2 \times n = n^3 \cong O(n^3)$$

```
2.sum:=0;
For k:=1 to n do
 For j:=1 to k do
  Sum:=sum+10;
```

$$\sum_{k=1}^{n}\sum_{j=1}^{k} 1 = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \cong O(n^2)$$

```
3.sum:=0;
For k:=1 to n do
 For j:=1 to sqr(k) do
   For m:=1 to j do
    Sum++;
```

$$\sum_{k=1}^{n}\sum_{j=1}^{k^2}\sum_{m=1}^{j} 1 = \sum_{k=1}^{n}\sum_{j=1}^{k^2} j = \sum_{k=1}^{n} \frac{k^2(k^2+1)}{2} \cong O(n^5)$$

(لأن المجموع يرفع الدرجة بمقدار واحد)

```
4. K=55;
For(i=0;i<n;i++)
   For (j=0;j<n;j++)
     K++;
```

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n = n \times n = n^2 \cong O(n^2)$$

```
5.procedure add_matrix(A,B,C)
For i:=1 to n
  For j:=1 to m
    C[i][j]=A[i][j]+B[i][j];
```

$$\sum_{i=1}^{n}\sum_{j=1}^{m} 1 = \sum_{i=1}^{n} m = m \times n \cong O(n^2)$$

ملاحظة:

$$\sum_{i=1}^{n} 2i-1 = 2\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1 = 2\ \frac{n(n+1)}{2} - n = n^2 + n - n = n^2 \cong O(n^2)$$

```
6.sum:=0;
For k:=1 to n do
  For j:=1 to sqr(k) do
   If(j mod k=0)then
    For m:=1 to j do
      Sum++;
```

$$\sum_{k=1}^{n}\sum_{j=1}^{k^2} ||\ \mathrm{j} = \sum_{k=1}^{n}(k+2k+3k+\cdots\ldots\ldots\ldots.+k.k) = \sum_{k=1}^{n} k(1+2+3+\cdots+k) =$$

$$\sum_{k=1}^{n} k\ \frac{k(k+1)}{2} \cong O(n^4)$$

أو (الشرط j mod k=0 يعني أن j=ck)

$$\sum_{k=1}^{n}\sum_{ck=1}^{k^2} ck = \sum_{k=1}^{n} k \sum_{c=1}^{k} c = \sum_{k=1}^{n} k\ \frac{k(k+1)}{3} \cong\ O(n^4)$$

(لأن المجموع يرفع الدرجة بمقدار واحد)

```
7.i:=1; t:=2;
  While(i<n)do
    Begin
      Sum:=sum*2*t;
       t:=t+2;
End;
```

احسب عدد عمليات الضرب و الجمع و درجة التعقيد:

عدد عمليات الضرب:

$$\sum_{i=1}^{n-1} 2 = 2(n-1) \cong O(n)$$

عدد عمليات الجمع:

$$\sum_{i=1}^{n-1} 1 = (n-1) \cong O(n)$$

```
8.procedure Ingress(n);
begin
     For i:=1 to n do
       Read(e[i];
           next i;
end;
```

$$\sum_{i=1}^{n} 1 = n \cong O(n)$$

9. s:=0;
For i:=1 to n do
 For j:=1 to i do
  S++;

$$\sum_{i=1}^{n}\sum_{j=1}^{i}1=\sum_{i=1}^{n}i=\frac{n(n+1)}{2}\cong O(n^2)$$

10. sum:=0;
 For i:=1 to n do
  For j:=1 to i^2 *do*
   Smm+=1;

$$\sum_{i=1}^{n}\sum_{j=1}^{i^2}1=\sum_{i=1}^{n}i^2=\frac{n(n+1)(2n+1)}{6}\cong O(n^3)$$

11.sum:=100;
 For i:=1 to sqr(n) do
  For j:=1 to i do
   Sum-:=1;

$$\sum_{i=1}^{n^2}\sum_{j=1}^{i}1\ =\sum_{i=1}^{n^2}i=\frac{n^2(n^2+1)}{2}\cong O(n^4)$$

12.for k:=1 to n do
 For j:=2 to k do
  Sum++;

$$\sum_{k=1}^{n}\sum_{j=2}^{k}1=\sum_{k=1}^{n}k-1=\sum_{k=1}^{n}k-\sum_{k=1}^{n}1=\frac{n(n+1)}{2}-n=$$
$$\frac{n^2+n-2n}{2}=\frac{n^2-n}{2}\cong O(n^2)$$

13.unsigned int sum(int n){
Unsigned int I,partial_sum=0;
for(i=1;i<=n;i++)
partial_sum+=i*i*I;
return(partial_sum);
}

$$\sum_{i=1}^{n}2=2n\cong O(n)$$

قوانين:

جمع n مرة

$$\sum_{i=1}^{n} i = 1+2+3+\cdots \ldots\ldots + n = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = 1+2^2+3^2+\cdots \ldots. + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2$$

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

عدد الأعداد الفردية:من 1 ⟵ n هي:

$$\sum_{i=1}^{n} 1 = \frac{n+1}{2}$$

عدد الأعداد الزوجية:

$$\sum_{i=1}^{n} 1 = \frac{n}{2} \qquad \equiv \qquad \sum_{i=1}^{n} \Big\|_{i \bmod 2=0} 1 = \frac{n}{2}$$

```
14.for i:=n-1 downto 1 do
     For j:=1 to I do
        If(a[j+1]<a[j])then
          Swap;
```

$$\sum_{i=1}^{n-1}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2} \cong O(n^2)$$

```
15.for i:=1 to n-1 do
      For j:=1 to n-1 do
         If(a[j]>a[j+1])then
            Swap;
```

$$\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} 1 = (n-1)(n-1) = n^2-2n+1 \cong O(n^2)$$

```
16. prod:=1;
     While(n>1)do
```

```
Begin
 M:=n;
  While(m>1)do
   Begin
     Prod:=prod*2; m:=m div 2;
     End;
 N:=n div 2;
  End;
```

في الحلقة الداخلية لدينا:

رقم الدخول	قيمة m
1	m
2	$\frac{m}{2}$
3	$\frac{m}{2^2}$
4	$\frac{m}{2^3}$
J+1	$\frac{m}{2^j}$

آخر مرة يدخل فيها الحلقة: $\frac{m}{2^j} = 2$

$$\log_2 \frac{m}{2^j} = \log_2 2 \underset{\text{ومنه}}{\Rightarrow} \log_2 m - \log_2 2^j = 1 \underset{\text{ومنه}}{\Rightarrow} \log_2 m - j = 1 \underset{\text{ومنه}}{\Rightarrow}$$

$$\mathrm{j} = \log_2 \mathrm{m} - 1 \cong \mathrm{O}(\log_2 \mathrm{m})$$

و كذلك الأمر بالنسبة للحلقة الخارجية:

رقم الدخول	قيمة m
1	n
2	$\frac{n}{2}$
3	$\frac{n}{2^2}$
4	$\frac{n}{2^3}$
J+1	$\frac{n}{2^j}$

آخر مرة يدخل فيها الحلقة: $\frac{n}{2^j} = 2$

$$\mathrm{j} = \log_2 \mathrm{n} - 1 \cong \mathrm{O}(\log_2 \mathrm{n})$$

فيكون تعقيد الخوارزمية ككل هي:

$$\mathrm{O}(\log_2 \mathrm{n} \times \log_2 \mathrm{m})$$

Prod:=$2*2^2*2^3*2^4*$...........$*2^j=2^{\frac{j(j+1)}{2}}$

و لدينا:

j$\cong \log_2 n = \log_2 2^k = k$

prod:=$2^{\frac{k(k+1)}{2}}$

```
17. prod:=1;
For k:=1 to n do
For j:=1 to sqr(n) do
 If(j mod sqrt(n)=0)then
     For m:=1 to j do
         Prod:=prod*2;
```

$$\sum_{k=1}^{n}\sum_{\substack{j=1 \\ J \bmod \sqrt{2}=0}}^{n^2}\left\|\sum_{m=1}^{j} 1 = \sum_{k=1}^{n}\sum_{j=1}^{n^2} \|j\right.$$

إن الشرط j mod $\sqrt{n}$=0 يعني أن j=c$\sqrt{n}$

$$\sum_{k=1}^{n}\sum_{c\sqrt{n}=1}^{n^2} c\sqrt{n} = \sum_{k=1}^{n}\sqrt{n}\sum_{c=1}^{n\sqrt{n}} c = \sum_{k=1}^{n}\sqrt{n}\,\frac{n\sqrt{n}(n\sqrt{n}+1)}{2} =$$

$$\sum_{k=1}^{n}\frac{n^2(n\sqrt{n}+1)}{2} = \frac{n^3(n\sqrt{n}+1)}{2} = \frac{n^4\,n^{\frac{1}{2}}+n^3}{2} \cong O(n^{4.5})$$

أو مباشرة:

$$\sum_{k=1}^{n}\sqrt{n}+2\sqrt{n}+3\sqrt{n}+4\sqrt{n}+\cdots..+n^2 = \sum_{k=1}^{n}\sqrt{n}\,\frac{n\sqrt{n}(n\sqrt{n}+1)}{2} \cong O(n^{\frac{9}{2}})$$

Prod:=$\left[2^{\sqrt{n}} * 2^{2\sqrt{n}} * 2^{3\sqrt{n}} * \ldots..* 2^{n^2}\right]^n$

Prod:=$2^{\sqrt{n}(1+2+3+\cdots.......+n\sqrt{n})^n}$

Prod:=$2^{n\sqrt{n}\left(\frac{n\sqrt{n}(n\sqrt{n}+1)}{2}\right)}$

```
18.prod:=1;
While(n>1)do
Begin
Prod:=prod*n; n:=n div 4;
End;
```

رقم الدخول	قيمة n
1	n
2	$\frac{n}{4}$
3	$\frac{n}{4^2}$
4	$\frac{n}{4^3}$
j+1	$\frac{n}{4^j}$

$\frac{n}{4^j} \leq 1 \underset{\text{ومنه}}{\Rightarrow} j = \frac{\log_2 n}{2} \underset{\text{ومنه}}{\Rightarrow} cost \cong O(\log_2 n)$

أو :

$$\frac{n}{4^j} = 2 \underset{\text{ومنه}}{\Rightarrow} \log_4 \frac{n}{4^j} = \log_4 2$$
$$\log_4 n - \log_4 4^j = \log_4 2 \underset{\text{ومنه}}{\Rightarrow} j = \log_4 n - \log_4 2 \cong O(\log_4 n)$$

Prod:=n* $\frac{n}{4} * \frac{n}{4^2} * \frac{n}{4^3}$ *.............. *$\frac{n}{4^{j-1}}$

Prod:=$2^k * \frac{2^k}{4^1} * \frac{2^k}{4^2} * \ldots\ldots\ldots\ldots\ldots * \frac{2^k}{4^{j-1}} = \frac{2^{k(j-1)}}{4^{\frac{j(j-1)}{2}}}$

$= \frac{2^{k(j-1)}}{2^{j(j-1)}} = \quad 2^{k(j-1)-j(j-1)}$

J=$\frac{\log_2 n}{2} = \frac{\log_2 2^k}{2} = \frac{k}{2}$

Prod:=$2^{\left(\frac{k}{2}-1\right)(k-\frac{k}{2})}$ = $2^{\left(\frac{k}{2}-1\right)(\frac{k}{2})}$

19.prod:=1;		prod:=1;
While(n>1)do	هذه الخوارزمية تكافئ ⟹	While(n>1)do
Begin		Begin
Prod:=Prod*n;		Prod:=Prod*n;
m:=n div 2:		n:=n div 4:

نفس السابقة نجد أن:

$$cost \cong O(\log_2 n)$$
$$j = \frac{\log_2 n}{2}$$

بافتراض أن $n=2^{2k}$:

$$j=\frac{\log_2 2^{2k}}{2}=\frac{2k}{2}=k$$

هذه الخوارزمية تكافئ

```
20.Prod:=1;
While(n>1)do
Begin m:=n;
  While(m>1)do
     Begin
       Prod:=prod*2;
       n:= m div 2;
```

```
Prod:=1;
While(n>1)do
Begin m:=n;
   While(m>1)do
      begin
         m:=m div 4;
          prod:=prod*4;
```

في الحلقة الداخلية لدينا:

رقم الدخول	قيمة m
1	m
2	$\frac{m}{4}$
3	$\frac{m}{4^2}$
j+1	$\frac{m}{4^j}$

$$\frac{m}{4^j}\le 1 \underset{\text{ومنه}}{\Rightarrow} j=\frac{\log_2 m}{2} O(\log_2 m)$$

و كذلك الأمر بالنسبة للحلقة الخارجية: $cost\cong O(\log_2 n)$

و يكون التعقيد ككل $cost\cong O(\log_2 n.\log_2 m)$
و ليس خطأ أن نكتب:

$$cost\cong O(\log_4 n.\log_4 m)$$

Prod:=$4 * 4^2 * 4^3 *\dots* 4^j= 4^{\frac{j(j+1)}{2}}$

J=k

Prod:=$4^{\frac{k(k+1)}{2}}=2^{k(k+1)}$

```
21. prod:=1;
While(n>1)do
Begin
Prod:=prod*n;
n:=sqrt(n);
end;
```

رقم الدخول	قيمة n
1	n
2	$n^{\frac{1}{2}}$
3	$n^{\frac{1}{2^2}}$
J+1	$n^{\frac{1}{2^j}}$

$$n^{\frac{1}{2^j}}=2$$

$$\frac{1}{2^j}\log_2 n = \log_2 2$$

$$\underset{\text{ومنه}}{\Rightarrow} \log_2 n = 2^j \;\rightarrow\; \log_2 \log_2 n = j \cong O(\log_2 \log_2 n)$$

```
22.Prod:=1;
While(n>1)do
Begin m:=n;
While(m>1)do
Begin prod:=prod*2; m:=√m end;
n:=√n;
end;
```

في الحلقة الداخلية لدينا:

رقم الدخول	قيمة m
1	m
2	$m^{\frac{1}{2}}$
3	$m^{\frac{1}{2^2}}$
j+1	$m^{\frac{1}{2^j}}$

$$m^{\frac{1}{2^j}} = 2$$

$$\frac{1}{2^j}\log_2 m = 1$$

$$j=\log_2 \log_2 m$$

وكذلك الأمر بالنسبة للحلقة الخارجية:

$$\log_2 \log_2 n$$

cost$\cong O(\log_2 \log_2 n \,.\log_2 \log_2 m)$

```
22. for i:=1 to n do
         If(i mod 3=0)then
            For j:=i to n do
             X:=x+3
Else
    For j:=1 to i do
       Y:=y*3;
```

عدد عمليات الجمع:

$$\sum_{i=1}^{n} \Bigg\| \sum_{j=i}^{n} 1$$

i mod 3 =0

ملاحظة: لدينا:

$$\sum_{j=1}^{n} 1 = \sum_{j=1}^{i-1} 1 + \sum_{j=i}^{n} 1$$

$$\sum_{j=i}^{n} 1 = \sum_{j=1}^{n} 1 - \sum_{j=1}^{i-1} 1$$

$$\sum_{i=1}^{n} \| n - i + 1$$

الشرط i mod 3=0 يعني i=3c:

$$\sum_{i=1}^{n} \Bigg\| n - \sum_{i=1}^{n} \Bigg\| i + \sum_{i=1}^{n} \| 1 =$$

$$\sum_{c=1}^{\frac{n}{3}} n - \sum_{c=1}^{\frac{n}{3}} 3c + \sum_{c=1}^{\frac{n}{3}} 1 = n \times \frac{n}{3} - 3\left(\frac{\frac{n}{3}\,(\frac{n}{3}+1)}{2}\right) =$$

$= \frac{n^2-2n^2-6n+n}{3} \cong O(n^2)$

عمليات الضرب:

$$\sum_{i=1}^{n} \Big\| \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} \| i$$

i mod 3 ≠ 0

مجموع الأعداد التي تقبل القسمة على 3 : و تساوي مجموع الأعداد من 1 ← n – مجموع الأعداد التي تقبل القسمة على 3:

$$= \frac{n(n+1)}{2} - 3 \times \frac{n}{3}\left(\frac{\frac{n}{3}+1}{2}\right) \cong O(n^2)$$

```
33. prod:=1;
 While(n>1)do
Begin
Prod:=prod*n;
n:=n div 2;
end;
```

احسب تعقيد هذه الخوارزمية في أسوأ الأحوال بدلالة (O).
بافتراض أن $n=2^k$ احسب قيمة prod بدلالة n أو K أو اثنيهما معاً.

رقم الدخول	قيمة n
1	n
2	$\frac{n}{2}$
3	$\frac{n}{2^2}$
j+1	$\frac{n}{2^j}$

آخر مرة يدخل فيها الحلقة هي n=2 :

$$\frac{n}{2^j} = 2 \rightarrow \log_2 \frac{n}{2^j} = \log_2 2$$
$$\log_2 n - \log_2 2^j = 1$$

و يمكن أن نكتب:

$$\frac{n}{2^j} \le 1 \rightarrow n \le 2^j \rightarrow \log_2 n = j \cong O(\log_2 n)$$

Prod:=n * $\frac{n}{2} * \frac{n}{2^2} * \frac{n}{2^3} * \ldots\ldots\ldots\ldots * \frac{n}{2^{j-1}}$

Prod:=$2^k * \frac{2^k}{2} * \frac{2^k}{2^2} * \frac{2^k}{2^3} * \ldots\ldots\ldots\ldots * \frac{2^k}{2^{j-1}} = \frac{2^{k(j-1)}}{2^{\frac{j(j-1)}{2}}} =$

Prod:=$2^{k(j-1)-\frac{j}{2}(j-1)} = 2^{(j-1)(k-\frac{j}{2})}$

j:=$\log_2 n = \log_2 2^k \rightarrow k \log_2 2 = j \rightarrow j = k$

prod:=$2^{(k-1)\left(k-\frac{k}{2}\right)} = 2^{(k-1)(\frac{k}{2})}$

24.for i:=1 to n-1 do

```
For j:=i+1 to n do
For k:=1 to j do
Begin
m:=m-1;
p:=p*4;
l:=l+55;
end;
```

الكلفة الأساسية هنا هي عملية الضرب:

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}1=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}j$$

ملاحظة:

$$\sum_{j=1}^{n}\mathrm{j}=\sum_{j=1}^{i}\mathrm{j}+\sum_{j=i+1}^{n}\mathrm{j}\rightarrow\sum_{j=i+1}^{n}\mathrm{j}=\sum_{j=1}^{n}\mathrm{j}-\sum_{j=1}^{i}\mathrm{j}$$

فيكون لدينا:

$$\sum_{i=1}^{n-1}\frac{n(n+1)}{2}-\frac{i(i+1)}{2}=$$

$$\sum_{i=1}^{n-1}\frac{n^2+n}{2}-\sum_{i=1}^{n-1}\frac{i^2+i}{2}=$$

$$\frac{1}{2}(n-1)(n^2+n)-\frac{1}{2}\sum_{i=1}^{n-1}i^2-\frac{1}{2}\sum_{i=1}^{n-1}i=$$

$$=\frac{1}{2}(n^3+n^2-n^2-n)-\frac{1}{2}\left(\frac{(n-1)(2n-1)n}{6}\right)-\frac{1}{2}\left(\frac{(n-1)n}{2}\right)=$$

$$\frac{4n^3-4n}{12}\cong O(n^3)$$

```
25. for i:=1 to n do
        if(odd(i))then
          for j:=1 to n do
            x:=x+1;
          for j:=1 to i do
            y:=y*2;
```

عدد عمليات الجمع و درجة التعقيد:

$$\sum_{\substack{i=1\\ i \bmod 2=1}}^{n}\left\|\sum_{j=1}^{n}1=\sum_{i=1}^{n}\right\|n=n*\frac{(n+1)}{2}\cong O(n^2)$$

عدد عمليات الضرب و درجة التعقيد:

$$\sum_{i=1}^{n}\left\|\sum_{j=1}^{i}1 = \sum_{i=1}^{n}\right\| i$$

مجموع الأعداد من 1 ← n – مجموع الأعداد الزوجية من 1 إلى n .

$$= \frac{n(n+1)}{2} - \frac{n}{2}\left(\frac{n}{2}+1\right) \cong O(n^2)$$

اكتب تابع sum_array لحساب مجموع عناصر نسق من الأعداد الصحيحة بشكل عودي.
اكتب إجرائية عودية bubble_sort (فرز الفقاعات).

Code

```
Function sum_array(a:mat;n:integer):integer;
Begin
If(n=1)then
Sum_array:=a[1]
Else
Sum_array:=a[n]+sum_array(a,n-1);
Procedure bubble_sort(var m:mat; n:integer);
Var temp,i:integer;
Begin
For i:=1 to n do {n-1}
If(m[i]>m[i+1])then
Begin
Temp:=m[i];
M[i]:=m[i+1];
M[i+1]:=temp;
End;
If(n>2)then
Sort(m,n-1);
End;
Begin
Writeln(sum_array(matrix,n));
Bubble_sort(matrix,n-1);
End.
```

Code

```
function pow2(x,n:integer):integer;
var t,res:integer;
begin
t:=x; res:=1;
while(n<>0)do
begin
if(n mod 2=1)then res:=res*t;
t:=t*t; n:=n div 2;
end;
pow2:=res;
end;
```

أسئلة عن تابعين power:

1 - ما الذي يحسبه كلا التابعين ؟
2 - أعط مثالاً عددياً يوضح آلية عمل كل تابع.
3 - قدر زمن تنفيذ كلا من التابعين.
4 - ما العلاقة بين هاتين التابعين.
5 - أي التابعين أفضل ؟ مع التبرير.

التابعان يحسبان x^n (رفع إلى قوة)

رقم الدخول	قيمة n
1	n
2	$\frac{n}{2}$
3	$\frac{n}{2^2}$
j+1	$\frac{n}{2^j}$

آخر مرة يدخل الحلقة n=1

$$\frac{n}{2^j} = 1 \rightarrow n = 2^j$$

$$\log_2 n = \log_2 2^j \rightarrow \log_2 n = \text{j} \log_2 2$$

$\text{j}=\log_2 n \cong O(\log_2 2)$

يفضل التابع التكراري لأنه لا يستغل ذاكرة كبيرة مثل العودية لأن العودية تعاني من الاستدعاءات العودية حيث يسبب كل استدعاء عودي نسخ التابع (متحولاته فقط) مما يسبب استدعاءات كبيرة و استهلاك الذاكرة.

عرف نمط المعطيات الأنسب لتمثيل الأعداد العقدية:
a+ib حيث i هي جذر العدد 1-
باستخدام هذا النمط عرف برامج جزئية تقوم بإجراء عمليات الجمع و الطرح و الضرب و القسمة على الأعداد العقدية (أربع برامج مختلفة).

Code

```
program test;
type
complex=record
re,im:real;
end;
var a,b,c:complex;
procedure sum_complex(a,b:complex);
var c:complex;
begin
writeln('enter the number complex of first');
writeln('enter part real');
readln(a.re);
writeln('enter part imagination');
readln(a.im);
writeln('enter the number complex of second');
writeln('enter part real');
readln(b.re);
writeln('enter part imagination');
readln(b.im);
c.re:=a.re+b.re;
c.im:=a.im+b.im;
writeln('(',a.re:4:2,'+',a.im:4:2,'i)+(',b.re:4:2,'+',b.im:4:2,'i)='
,c.re:6:2,' ','+',c.im:6:2,'i');
end;
procedure sub_complex(a,b:complex);
var c:complex;
begin
writeln('enter the number complex of first');
writeln('enter part real');
readln(a.re);
writeln('enter part imagination');
readln(a.im);
writeln('enter the number complex of second');
writeln('enter part real');
readln(b.re);
writeln('enter part imagination');
readln(b.im);
c.re:=a.re-b.re;
c.im:=a.im-b.im;
writeln('(',a.re:4:2,'+',a.im:4:2,'i)-(',b.re:4:2,'+',b.im:4:2,'i)='
,c.re:6:2,' ','+',c.im:6:2,'i');
end;
procedure mult_complex(a,b:complex);
begin
```

```
writeln('enter the number complex of first');
writeln('enter part real');
readln(a.re);
writeln('enter part imagination');
readln(a.im);
writeln('enter the number complex of second');
writeln('enter part real');
readln(b.re);
writeln('enter part imagination');
readln(b.im);
c.re:=(a.re*b.re)-(a.im*b.im);
c.im:=a.re*b.im+a.im*b.re;
writeln('(',a.re:4:2,'+',a.im:4:2,'i)*(',b.re:4:2,'+',b.im:4:2,'i)='
,c.re:6:2,' ','+',c.im:6:2,'i');
end;
procedure div_complex(a,b:complex);
var c:complex;  i:real;
begin
writeln('enter the number complex of first');
writeln('enter part real');
readln(a.re);
writeln('enter part imagination');
readln(a.im);
writeln('enter the number complex of second');
writeln('enter part real');
readln(b.re);
writeln('enter part imagination');
readln(b.im);
i:=(b.re*b.re)+(b.im*b.im);
c.re:=(a.re*b.re-a.im*-b.im)/i;
c.im:=(a.re*-b.im+a.im*b.re)/i;
writeln('(',a.re:4:2,'+',a.im:4:2,'i)/(',b.re:4:2,'+',b.im:4:2,'i)='
,c.re:6:2,' ','+',c.im:6:2,'i');
end;
begin
sum_complex(a,b);
sub_complex(a,b);
mult_complex(a,b);
div_complex(a,b);
readln;
end.
```

صورة من تنفيذ البرنامج السابق:

```
enter the number complex of first
enter part real
5
enter part imagination
3
enter the number complex of second
enter part real
1
enter part imagination
2
(5.00+3.00i)+(1.00+2.00i)=  6.00 +  5.00i
enter the number complex of first
enter part real
5
enter part imagination
3
enter the number complex of second
enter part real
1
enter part imagination
2
(5.00+3.00i)-(1.00+2.00i)=  4.00 +  1.00i
enter the number complex of first
enter part real
5
enter part imagination
3
enter the number complex of second
enter part real
1
enter part imagination
2
(5.00+3.00i)*(1.00+2.00i)= -1.00 + 13.00i
enter the number complex of first
enter part real
5
enter part imagination
3
enter the number complex of second
enter part real
1
enter part imagination
2
(5.00+3.00i)/(1.00+2.00i)=  2.20 + -1.40i
```

ليكن لدينا البرنامج التالي:

Code

```
program test;
function ingress(x,y:integer):integer;
begin
if(y=0)then
ingress:=0
else
ingress:=ingress(y-1,x)+x;
end;
begin
writeln(ingress(5,2));
readln
end.
```

1 - وضح آلية استدعاء التابع و ما هو خرج البرنامج.
2 - ما هو عمل التابع.
3 - ماذا يحدث عند الاستدعاء ingress(-2,-2) و لماذا.
4 - هل الاستدعاءين التاليين مقبولين ingress(2,-2) أو ingress(-2,2); و ما هو خرج التابع

```
ingress( 5,2);
ingress (1,5)+5 ↑→ =5+5=10
ingress(4,1)+1  |→ =4+1=5
ingress(0,4)+4  |→ =0+4=4
   ↙
ingress:=0
```

خرج البرنامج هو 10.
عمل التابع هو ضرب عددين صحيحين.
(يجب أن يكون القيمتان المدخلتين معاً أكبر تماما من الصفر) و إلا يحدث overflow لأن عند الاستدعاء ingress(-2,-2) لا يوجد شرط لتوقف العودية.
و بالتالي أي استدعاء من الشكل ingress(x,y):x,y<0 مرفوض.
الاستدعائين ingress(2,-2) و ingress(-2,2) مقبولين لأن يوجد شرط يوقف العودية.
خرج التابع هو -4.
إذا عمل التابع هو ضرب عددين صحيحين بحيث أي استدعاء من الشكل ingress(x,y):x and y<0 مرفوض.

ليكن لدينا البرنامج التالي:

Code

```
function fun(x,y:integer):integer;
begin
if(y=0)then
fun:=x
else
fun:=fun(y-1,x-1)+1;
end;
```

ما هو خرج التعليمة التالية: writeln(fun(3,2))

a. 2
b. 1
c. **3**
d. 6
e. الجواب الصحيح ليس مما سبق.

```
Fun(3,2);
1+fun(1,2); ↑1+2=3
1+fun(1,0); |1+1=2
   ↙
Fun:=1
```

ما هو خرج التعليمة التالية: writeln(fun(100,2))

a. 2
b. 1
c. **100**
d. 6
e. الجواب الصحيح ليس مما سبق.

```
Fun(100,2);
1+fun(1,99); ↑1+99=100
1+fun(98,0); |1+98=99
   ↙
Fun:=98
```

ما هو خرج التعليمة التالية: writeln(fun(6,3))

a. 2
b. 1
c. 3
d. 6
e. الجواب الصحيح ليس مما سبق.

الجواب : يحدث overflow.

من أجل البرنامج التالي وضح آلية الأستدعاء وخرج البرنامج:

```
program test;
procedure sum(x,y:integer; var res:integer);
begin
if(x=0)then
res:=y
else
begin
y:=y+1;
sum(x-1,y,res);
res:=x+y;
end;
end;
var res:integer;
begin
sum(4,3,res);
writeln(res);
readln
end.

Sum(3,5);    res:=3+5=8
Sum(2,6);    res:=2+6=8
Sum(1,7);    res:=1+7=8
Sum(0,8);
   ↙
  Res:=8
```

خرج هو 8

ما هو خرج البرنامج التالي:

```
program test;
procedure ingress(x,y:integer; var res:integer);
begin
if(x=0)then
res:=y
else
begin
ingress(x-1,y,res);
res:=res+y;
ingress(x-1,y,res);
res:=res+1;
end;
end;
```

```
function fun(x,y:integer):integer;
begin
if(y=0)then
fun:=1
else
begin
fun:=fun(x,y-1);
fun:=1+fun(x,y-1);
end;
end;
var res:integer;
begin
ingress(2,5,res);
writeln(res);
writeln(fun(2,3));
readln
end.
```

خرج البرنامج هو:

7
4

ما هو خرج البرنامج التالي:

```
program test;
function ingress(x,y:integer):integer;
begin
if(x<=0)then
ingress:=x+1+y div 2
else
ingress:=x+y div 2+ingress(x-1,y);
end;
begin
writeln(ingress(ingress(2,2),2));
writeln(2,ingress(2,2));
writeln(ingress(2,ingress(2,2)));
readln;
end.
```

Solutions :
Outputs:
37
27
13

اكتب خرج البرنامج التالي:

```
program test;
function ingress(x,y:integer):integer;
begin
if(x<=0)then
ingress:=x+1+y div 2
else
ingress:=x+y div 2+ingress(x-1,y*x div 10);
end;
```

```
begin
writeln(ingress(ingress(2,2),2));
writeln(2,ingress(2,2));
writeln(ingress(ingress(2,ingress(2,2)),20));
writeln(ingress(6,20));
readln;
end.
```

Solution :
Outputs:
17
25
42
42

ليكن لدينا:

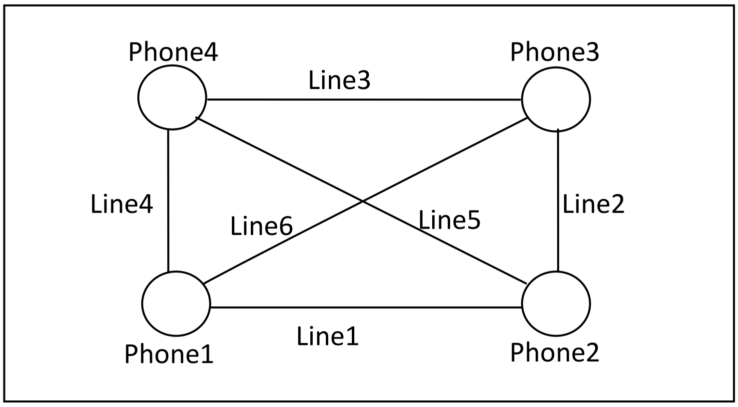
```
program test;
var x,y:integer;
procedure ingress(x,y:integer);
var i:integer;
begin
i:=1;
x:=x xor y;
y:=x xor y;
x:=x xor y;
repeat
x:=x+i div y+x mod y;
y:=y+x+ y mod x;
i:=i+1;
until(i>4);
writeln(x,',',y);
end;
begin
ingress(4,5);
readln;
end.
```

خرج البرنامج هو:

a. **48,112**
b. 64,144
c. 24,56
d. 12,28
e. None of the above

ترغب شركة بربط هواتفها بعضها ببعض مباشرة دون استخدام أي مقسم هاتف فعلى سبيل المثال لا الحصر و كما يبين الشكل من أجل عدد هواتف nphone=4 فإن عدد الخطوط التي تربط كافة الخطوط هي nline=6.

Telephone Network



اكتب بلغة turbo pascal برنامج يقوم بـــــــــــــــــــ:

- بقراءة عدد الهواتف nphones من لوحة الملامس.
- كتابة تابع calculatenbline تمرر إليه كوسيط عدد الهواتف nphones ويرد عدد الخطوط الرابطة nline.
- كتابة إجرائية printnetwork تمرر إليه عدد الهواتف فيطبع كافة الخطوط الواصلة بين الهواتف.
- مثال: من إجل nphone=4 فيتم طباعة:

line 1 to 2
line 1 to 3
line 1 to 4
line 2 to 3
line 2 to 4
line 3 to 4

Code

```
PROGRAM test;
const max=100;
var
nphones,i:integer;
mat:array[1..max] of integer;
function calculatenblines(nphones:integer):integer;
begin
calculatenblines:=nphones*(nphones-1) div 2;
end;
procedure printnetwork(nphones:integer);
var i,j:integer;
begin
for i:=1 to nphones-1 do
for j:=i+1 to nphones do
writeln('line ',mat[i],' to ',mat[j]);
end;
```

```
begin
readln(nphones);
for i:=1 to nphones do
mat[i]:=i;
writekn(' number of lines in networks = ',calculatenblines(nphones));
printnetwork(nphones);
readln;
end.
```

خرج البرنامج من أجل nphones = 4 هو :

```
4
number of lines in networks = 6
line 1 to 2
line 1 to 3
line 1 to 4
line 2 to 3
line 2 to 4
line 3 to 4
```

المهندس خالد ياسين الشيخ

خرج البرنامج من أجل nphones=10 هو :

```
5
number of lines in networks = 10
line 1 to 2
line 1 to 3
line 1 to 4
line 1 to 5
line 2 to 3
line 2 to 4
line 2 to 5
line 3 to 4
line 3 to 5
line 4 to 5
```

المهندس خالد ياسين الشيخ

اكتب تابع يقوم بعكس خانات عدد صحيح مثال لدينا العدد 123 يقوم التابع بعكس أرقامه فيكون 321 العدد 9812- يقوم بعكس أرقامه 2189-.
و المطلوب تابعين أحدهما تكراري و الآخر عودي.

```
Code
PROGRAM test;
function reverse_number(n:integer):integer;
var res:integer;
begin
res:=0;
while(n<>0)do
begin
res:=res*10+n mod 10;
n:=n div 10;
end;
```

```
reverse_number:=res;
end;
function rev_num(n,res:integer):integer;
begin
if(n=0)then
rev_num:=res
else
begin
res:=res*10+n mod 10;
rev_num:=rev_num(n div 10,res);
end;
end;
var n:integer;
begin
repeat
writeln('enter the (0 to stop) number');
readln(n);
if(n<>0)then
begin
writeln('reverse ',n,' = ',reverse_number(n));
writeln('reverse ',n,' = ',rev_num(n,0));
end;
until(n=0);
end.
```

خرج البرنامج السابق شبيه بالتالي

```
enter the (0 to stop) number
123
reverse 123 = 321
reverse 123 = 321
enter the (0 to stop) number
-123
reverse -123 = -321
reverse -123 = -321
enter the (0 to stop) number
325
reverse 325 = 523
reverse 325 = 523
enter the (0 to stop) number
10236
reverse 10236 = -2335
reverse 10236 = -2335
enter the (0 to stop) number
-5894
reverse -5894 = -4985
reverse -5894 = -4985
enter the (0 to stop) number
456
reverse 456 = 654
reverse 456 = 654
enter the (0 to stop) number
92
reverse 92 = 29
reverse 92 = 29
enter the (0 to stop) number
```

المهندس خالد ياسين الشيخ

Code

إجرائيات و توابع تكرارية تحول من عشري إلى ثنائي ومن ثنائي إلى عشري.

```
program test;
type
mat=array[1..30] of 0..1;
var a:mat;
function pow(x:integer):integer;
var c,i:integer;
begin
c:=1;
for i:=1 to x do
c:=c*2;
pow:=c;
end;
```

إجرائية تكرارية تحول من عشري إلى ثنائي:

```
procedure Bin(x:integer);
var i,k:integer;
begin
i:=0;
while(x<>0)do
begin
i:=i+1;
a[i]:=x mod 2;
x:=x div 2;
end;
for k:=i downto 1 do
write(a[k]);
end;
```

إجرائية عودية تحول من عشري إلى ثنائي:

```
procedure bin1(x:integer;var i:integer);
var k:integer;
begin
if(x<>0)then
begin
i:=i+1;
a[i]:=x mod 2;
bin1(x div 2,i);
end
else
for k:=i downto 1 do
write(a[k]);
end;
```

إجرائية تكرارية تحول من ثنائي إلى عشري:

$$(\overset{2}{1}\overset{1}{0}\overset{0}{1})_2 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 = 1 + 0 + 4 = (5)_{10}$$

```
function dec(x:integer):integer;
var res,i:integer;
```

```
begin
res:=0; i:=0;
while(x<>0)do
begin
res:=res+x mod 10*pow(i);
x:=x div 10;i:=i+1;
end;
dec:=res;
end;
```

إجرائية عودية تحول من ثنائي إلى عشري:

```
function dec1(x,res,i:integer):integer;
begin
if(x<>0)then
begin
res:=res+x mod 10*pow(i);
dec1:=dec1(x div 10,res,i+1);
end
else
dec1:=res;
end;
var i:integer;
begin
i:=0;
bin(2);
writeln;
bin1(2,i);

writeln;
writeln(dec(1001));
writeln(dec1(11111,0,0));
readln
end.
```

صورة من نتيجة تنفيذ البرنامج السابق:



اكتب برنامج يقوم بحساب عدد الأحرف و عدد الكلمات من خلال نص مكتوب باللغة الإنكليزية حيث يتم إدخاله من جهاز الدخل القياسي (لوحة المفاتيح) على شاشة الخرج القياسية (الشاشة) علماً أن الفاصل بين الكلمة و الأخرى هو فراغ واحد على الأقل و النص ينتهي بنقطة (.) :

مثال لنفرض=لتخيل اننا أدخلنا النص التالي:

```
Hi how are you
I am from Syrian          arab              republic
syria                   arabic


Good bye.
```

ملاحظة: يجب معالجة جميع حالات الإدخال (فراغات ،أسطر فارغة).

عندها يجب أن يكون خرج البرنامج هو عدد الأحرف في النص= 54

و عدد الكلمات في النص = 14
ملاحظة:سوف أجعل المتحول r يحوي عدد الأحرف و المتحول w يحوي عدد الكلمات.

وسوف أحل هذه المسالة بطريقتين:

الطريقة التي تخطر على بال المبرمج المبتدئ وهي الطريقة المعقدة (من حيث غموض الخوارزمية (حل عشوائي))و هي كالتالي:

Code

```
program test;
var c:char;
w,r,m:integer;
ok:boolean;
begin
r:=0;
w:=0;
m:=0;
ok:=false;
while(true)do
begin
if(c='.')then
break;
read(c);
if (ok=false)and((c=chr(13)) and(r<>0)) then
begin
w:=w+1;
ok:=true
end
else
if(r<>0)and(c='.') then
if ok=false then
w:=w+1;
if(c=' ') then
begin
ok:=false;
if(r<>0)then
w:=w+1;
while(c=' ') do
read(c);
end;
case c of
'A'..'Z','a'..'z','0'..'9':begin r:=r+1; ok:=false; end;
end;
case ord(c) of
97..122:m:=m+1;
end;
end;
if(r<>0)then
begin
writeln('numbers of characters in text = ',r);
writeln('numbers of words in test = ',w);
```

```
writeln(m);
end
else
writeln('empty text');
readln;
end.
```

صورة من تنفيذ البرنامج بفرض أننا أدخلنا النص السابق:

```
Hi how are you
I am from syrian              arab           republic
syria                 arabic




Good bye.
numbers of characters in text = 54
numbers of words in test = 14
```

عدل البرنامج السابق بحيث يحسب عدد الحروف الصغيرة و الكبيرة و الأرقام المكونة في النص المدخل.

الطريقة الثانية و هي الطريقة النظامية بالحل نتعامل مع الشاشة تماما كما نتعامل مع الملف :

Code

```
program test;
var c:char;
w,r:integer;
ok:boolean;
s:string;
begin
r:=0;
w:=0;
while(not eof)do
begin
s:='';
ok:=false;
while(not eoln)do
begin
read(c);
if c in ['a'..'z','A'..'Z','0'..'9'] then
begin
r:=r+1;
ok:=true;
s:=s+c;
end;
if ((s<>'')and(c=' '))or (eoln)then
begin
if ok then
w:=w+1;
s:='';
```

```
end;
if(c='.')then
break;
end;
if(c='.')then
break;
readln;
end;
if(r<>0)then
begin
writeln('numbers of characters in text = ',r);
writeln('numbers of words in text = ',w);
end
else
writeln('empty text');
readln;
end.
```

صورة من تنفيذ البرنامج :

```
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
Hi how are you
Syria arabic


syrian arab republic


khaled yassin alsheikh  arabic arabic arabic



syria                                                syria
        syria arabic
                                  arab syrian



good bye.
numbers of characters in text = 116
numbers of words in text = 23
```

Syrian Arab Republic

بنى التحكم و المعطيات البسيطة والمعطيات المركبة

أنواع المعطيات البسيطة في لغة تربو باسكالturbo Pascal Simple Data Types:

تتضمن لغة باسكال أنواع معطيات متنوعة و غنية جعلت منها لغة سهلة في تمثيل أنواع المعطيات المختلفة و بأفضل طريقة بالإضافة إلى أنها تُمكن من إنشاء أنواع المعطيات التي نريد و حسب الحاجة التي يقتضيها البرنامج .

تزودنا أنواع المعطيات data types بقوالب من اجل تخزين المعلومات فيها فعلى سبيل المثال :نوع المعطيات عدد صحيح integer يُخزن العدد ضمن حجرتين من حجرات الذاكرة و بشكل مشابه فإن نوع المعطيات نسق array و الذي يخزن قيماً مختلفة تنتمي لنفس نوع المعطيات يضع عناصره في في حجرات ذاكرة عددها حسب نوع و عدد عناصر هذا النسق.

فإذا كان لدينا نسق مكون من 100 عدد صحيح فسوف يخصص لهذ القيم 200 بايت(حجرة). من الذاكرة من أجل تخزينها .

يحدد نوع المتحولات كمية حجرات التخزين اللازمة للمتحولات و أنواع العمليات التي يمكن إجراؤها على هذه المتحولات و هما أمران هامان من أجل المترجم compiler و المبرمج على حد سواء.

لقد قامت لغة تربو باسكال بتوسيع لغة باسكال وزودتها بمجموعة أكبر و أسهل من أنواع المعطيات و جعلت من لغة باسكال أكثر مرونة و سهلة في الاستخدام و تسمح لغة باسكال بتمثيل أنواع المعطيات البسيطة و المركبة.

البرامج بلغة باسكال Pascal Programs:

إن العنصر الأساسي الذي تتكون منه برامج باسكال هو العبارة statement و يتم تنفيذ البرنامج عبارة عبارة حيث تمثل العبارات التعليمات المستقلة المطلوبة لإنجاز مهمة البرنامج فعلى سبيل المثال :نحن نستخدم عبارة الإلحاق من أجل تخزين المعلومات في متحولات البرنامج و نستخدم أيضاً عبارات استدعاء الإجراءات و التوابع من أجل حساب قيم أو إنجاز عمل مطلوب.

هناك عدة اعتبارات مسموح بها عند كتابة العبارات منها أننا نستطيع ترك الفراغات المرغوبة بين عناصر العبارات كما نريد ونستطيع أيضاً أن نضع عناصر العبارة على أسطر مختلفة ولكن الشيء الوحيد الذي لا يمكن فعله أثناء كتابة العبارة هو وضع فراغات ضمن الكلمات أو السلاسل الرمزية strings فعلى سبيل المثال تُظهر لنا الأسطر التالية ثلاث طرائق مسموح بها في كتابة نفس العبارة:

```
Myint    :=7;   (*طريقة أولى*)
Myint:=7;       (* طريقة ثانية*)
Myint           (* طريقة ثالثة*)
:=

7;
```

لاحظ أن العبارة يمكن أن تُجزأ إلى أسطر متعددة كما هو الحال في الطريقة الثالثة من المثال السابق أما العبارة التالية فهي غير صحيحة بسبب وجود فرغ ضمن اسم المتحول:

```
My   int:=7;      (*عبارة خاطئة*)
```

هذه المرونة مفيدة جداً لأنها تسمح لنا باستخدام الفراغات كما نريد عند تنظيم برامجنا.

لكن عندما يقوم المترجم بقراءة الشفرة المصدرية source code فهو يحتاج إلى بداية كل عبارة ونهايتها ولهذا تزودنا الفاصلة المنقوطة ; بإمكانية الفصل بين العبارات حيث تأتي الفاصلة المنقوطة في نهاية كل عبارة ضمن البرنامج أو الإجراء أو التابع و لكن هناك عبارات لا تحتاج إلى فاصلة منقوطة مثل العبارة الأخيرة في نهايتها مثل العبارة الأخيرة في كل من البرنامج الرئيسي و الإجراء و التابع

يتضمن البرنامج التالي عدة مواضع تكون الفاصلة المنقوطة فيها اختيارية :

Code
program test; var intval:integer; procedure Awaituser; begin write('press enter to continue.'); readln   (‘;   can be omitted here’); end; procedure getinteger(message:string; var value:integer); begin write(message,' '); readln(value)   (‘;   can be omitted here’); end;

```
begin
getinteger('value?',intval);
writeln(intval);
Awaituser;
if(5>10) or (20+2>10 div 2)then
begin
writeln((5>10) or (20+2>10 div 2));
writeln(5=5 div 10)   (‘;   can be omitted here’);
end;
Awaituser         (‘;   can be omitted here’);
End.
```

رأينا في المثال السابق أن الفاصلة المنقوطة يمكن لأن تُهمل قبل المميز المحجوز END الذي ينهي البرنامج الرئيسي و الإجرائين Awaituser و Getinteger. ويمكن ان تهمل الفاصلة المنقوطة بعد آخر عبارة ضمن العبارة المركبة (تذكر أن العبارة المركبة تكون محاطة بالمميزين begin و end) هذا يعني أننا نستطيع تجاهل الفاصلة المنقوطة قبل end التي تنهي العبارة المركبة .

خرج البرنامج السابق هو:

```
value? 123
123
press enter to continue.
TRUE
FALSE
press enter to continue.
```

المهندس خالد ياسين الشيخ

التعليقات comments:

يمكننا أن نضيف إلى برامجنا بعض الملاحظات أو التعليقات و التي تساعدنا في فهم و تعديل و إصلاح برامجنا عند الحاجة. تملك لغة باسكال طريقيتين في تعيين التعليقات إذ يمكننا وضع التعليق الذي نريد ضمن القوسين (* comments *) أو ضمن القوسين {comments} فعندما يصادف المترجم القوس *) أو القوس } يتجاهل أي نص مكتوب بعده حتى يصل إلى الأقواس المقابلة لها أي (* أو } .

و البرنامج التالي يُظهر لنا مالذي يحدث إذا لم ننتبه إلى وضع الأقواس بالشكل الصحيح (نفذ البرنامج خطوة خطوة بالضغط على المفتاح الوظيفي F7).

Code
program test;
begin
writeln('one'); (* start a comment
writeln('Two');   finish a comment *)
writeln('three');
readln;
end.

بدأ التعليق في البرنامج السابق بعد أول استدعاء للإجراء writeln و انتهى عند آخر السطر التالي له و الذي حصل أننا أردنا أن نزود العبارتين الأولى و الثانية بتعليق مناسب لكل منهما و لكن المترجم قرأ السطر الثاني كاملاً واعتبره جزءاً من التعليق و كنتيجة لذلك كان خرج البرنامج يحتوي على الكلمتين one و three و لم يتضمن الكلمة tow.

خرج البرنامج السابق هو:

```
one
three
```

و البرنامج التالي يتضمن الطريقة الصحيحة في كتابة التعليقات للسطرين الأول و الثاني :

Code
program test; begin writeln('one'); (* start a comment*) writeln('Two');(* start and finish a comment *) writeln('three'); readln; end.

خرج البرنامج السابق هو:

```
one
Two
three
```

المهندس خالد ياسين الشيخ

يمكننا وضع تعليقاتنا في أي مكان ضمن البرنامج ما عدا وسط (ضمن) الأسماء أو السلاسل الرمزية مثل: أسماء الإجراءات أو أسماء المتحولات.

فعلى سبيل المثال السطر التالي سوف يقودنا إلى خطأ:

(* you cannot have   (*   nested *)   comments *)

لأن المترجم عندا يسير عبر التعليق يقوم ببساطة بالبحث عن رمز نهاية التعليق و هذا يعني أن التعليق بدأ بالرمز *) و المترجم يبحث عن الرمز المقابل له أي (* أما إذا بدأ التعليق بالرمز } فإن المترجم يبحث عن الرمز المقابل له {.

و في حال عثور المترجم على هذا الرمز يفترض أن التعليق قد انتهى و يستأنف ترجمة البرنامج و في المثال السابق يحاول المترجم تنفيذ الكلمة التالية:

comments   *)

و هذا سوف يؤدي إلى خطأ في عملية الترجمة.

و لكننا يمكننا أن نجعل المترجم يتقبل التعليقات المتداخلة بأن نستخدم الرمزين في بناء التعليق لنضع تعليقاتنا المتداخلة فلكي نضع تعليقاً ضمن آخر نستخدم أحد الرمزين من أجل التعليق الخارجي و الرمز الآخر يستخدم من أجل التعليق الداخلي(المتداخل) و المثال التال يُظهر تعليقاً متداخلاً و مقبولا من قبل المترجم :

(*   you can have   { quasi-nested   }   comments *)

{   you can have   (* quasi-nested *)   comments }

لاحظ كيفية وضع عبارات التعليق

Code
program test; procedure proc(var n:integer);forward; {prototype} function func(n:integer):integer; begin proc(n); func:=n; end; (* end function {return value } value=n*) procedure proc; begin n:=n div 10 div 2 div 5; end; begin   {main program } (*first statement *)writeln(func(200)); readln; end.

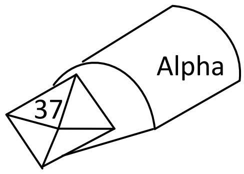
خرج البرنامج السابق هو:

```
2
```

المتحولات variables:

نحن نحتاج في البرامج التي نقوم بكتابتها بلغة برمجة معينة إلى تخزين عناصر متنوعة من المعلومات و معظم هذه العناصر تتغير قيمتها أثناء تنفيذ البرامج لذلك يجب على البرنامج أن يزودنا بطريقة لحفظ و تخزين هذه القيم و تدعى هذه الطريقة باستخدام المتحولات أو المتغيرات variables.

و المتحولات هي أسماء يُحجز لها مساحات تخزينية في الذاكرة من أجل تخزين معلومات فعندما نكتب برنامج ما لا و نصرح عن المتحولات فيه نقوم بحجز مساحة من الذاكرة مخصصة لهذه المتحولات و عندما نصرح عن متحول حددنا امه و نوهع يخصص المترجم مساحة من الذاكرة و يربط هذا الموقع من الذاكرة باسم المتحول و موضع التخزين هذا معروف فقط من قبل المترجم و لا حاجة إلى أن نعرفه و عندما نريد أن نقرأ المعلومات المخزنة في هذا الموقع أو نكتب شيئاً هناك علينا أن نضع اسم المتحول و يمننا أن نفترض=نتخيل المتحولات و أماكن تخزينها كصناديق للبيد الإلكتروني إذ يمثل اسم المتحول عنوان صندوق البريد و تمثل قيمة المتحول قيمة الرسالة (محتويات صندوق البريد) كما في التمثيل التخيلي=المنطقي التالي:



تسمية المتحولات naming variables:

يجب أن يبدأ تسمية الأسماء names أو المميزات identifiers بحوف أي أحد الحروف الصغيرة من a...z أو أحد الحروف الكبيرة من A...Z أو الرمز _ و من ثم يُتبع هذا الحرف بعدد من الأحرف أو الأرقام من 9 ...0 و لايسمح بوجود فراغات ضمن اسم المتحول و يمكن للمميز أن يكون بأي طول نريد و لكن المترجم يقرأ أول 63 رمزاً و فيما يلي بمض المميزات المسموح بها في لغة التربو باسكال وبعضها الآخر غير مسموح به:

- المميزات المسموح بها:

Q	LikeWise3	Avalidname
Info_sys_	Data	information

- المميزات غير المسموح بها:

	السبب ↓
Not-valid	(إشارة الناقص)
3LikeWise	(الرمز الأول رقم)
Info sys	(الفراغ)

لا تشكل حالة الأحرف الكبيرة والصغيرة أية أهمية في لغة باسكال حيث تُعامل الأحرف على انها حالة واحدة و هكذا فإن الأسماء التالية تعتبر متكافئة بلغة باسكال:

- vERSIOn1
- vErsiOn1
- Version1

التصريح عن المتحولات Variables Declarations:

تعتبر لغة باسكال شديدة الارتباط بأنواع المعطيات و هذا يعني أن المترجم يجب أن يعرف نوع المعطيات الواجب تخزينها كمتحولات قبل أن يستخدمها ضمن البرنامج ومن أجل إعطاء كل متحول نوع معطيات يجب التصريح declare عن هذا

المتحول أولاً فعندما نصرح عن متحول فإننا نعطي لهذا المتحول أسماً و نحدد له نوعاً و بنفس الطريقة يمكن التصريح عن عدة متحولات و يجب أن يبدأ التصريح عن المتحولات بالكلمة Var .

مثال:

```
Var max,min:real;
    Count:integer;
```

في التصريحات السابقة صرحنا عن ثلاثة متحولات حيث صرحنا عن متحولين من النوع الحقيقي real و صرحنا عن متحول من النوع الصحيح.

و إذا أردنا أن نصرح عن أكثر من متحول لهم نفس المعطيات يجب أن نكتب أسماء هذه المتحولات بشكل متتالي مع الفصل بين هذه الأسماء بواسطة الفاصلة , و إلحاق هذه الأسماء بنوع معطيات معين و الصيغة الكتابية للتصريح عن المتحولات كالتالي:

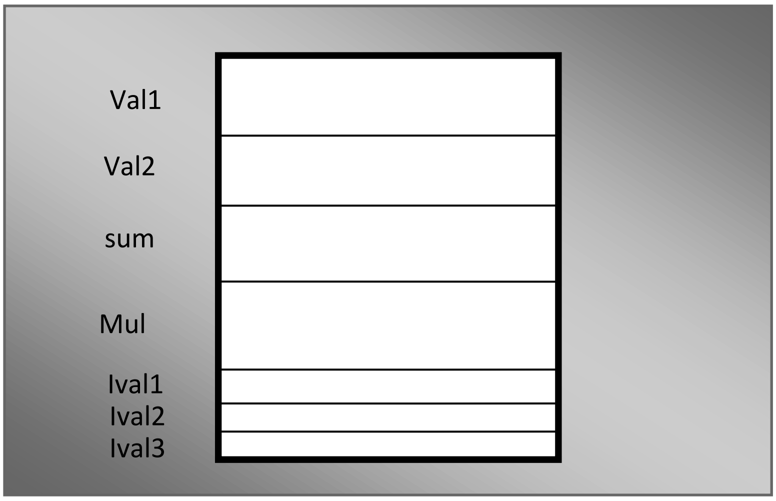
Var <اسم متحول 1> ,<اسم متحول 2>,........<اسم متحول n>:<نوع المعطيات>

و يجب أن لا ننسى فاصلة منقوطة ; بعد التصريح لنفصله عن التصريح التالي له و يمكننا ترك الفراغات التي نريدها بين أسماء المتحولات.

مثال:

```
Var
Val1,  val2
Sum,  Mul     :           real;
Ival1    ,   Ival2  , Ival3    :   integer     ;
```

عندما نصرح عن متحول فإن المترجم يُخصص موضعاً من الذاكرة لتخزين قيمة هذا المتحول و حجم موضع التخزين يعتمد على نوع المعطيات التي ينتمي إليها المتحول كما في التمثيل المنطقي=التخيلي :



بعد التصريح عن المتحولات يمكننا استخدامها و إلحاق قيم بها ضمن البرنامج كما نريد :

```
Val1:=5.0;
Val2:=2.0;
Sum:=val1+val2;
Mul1:=val1*val2;
IVal1:=2;
Ival2:=Ival1;
Ival3:=Ival1-Ival2;
```

بعد أن تنفذ هذه العبارات سوف يغدو حيز التخزين المخصص لهذه المتحولات محتوياً القيم التالية :

Val1	5.0
Val2	2.0
sum	7.0
Mul	10.0
Ival1	2
Ival2	2
Ival3	0

لا (تفترض= تتخيل) أبداً عند التعامل مع المتحولات أن المتحول يأخذ قيمة افتراضية ما.

المعاملات operators:

تملك لغة باسكال مجموعة كبيرة من أنواع المعطيات بالإضافة إلى أنها تمكننا من إنشاء أنواع المعطيات الخاصة بنا و المعامل عبارة عن عملية تطبق على القيم لإنتاج قيمة جديدة و المعامل بشكل عام يأخذ المعلومات في شكل معين و يعيدها في شكل مختلف فعلى سبيل المثال معامل الجمع + يأخذ المعلومات في شكل قيمتين و يعيدها على شكل مجموع هاتين القيمتين مثل هذا المعامل يأخذ قيمتين و يعيد قيمة و يدعى بالمعامل الثنائي binary operator و القيمة انتعلقة بالمعامل تعرف بالحد operand و المعاملات و الحدود يشكلون التعبير expression فعندما نكتب تعبيراً مثل:

55 + myval;

فالعدد 55 يكون الحد اليساري left operand لعملية الجمع و المتحول myval هو الحد اليميني right operand و عملية + هو المعامل و هكذا فإن المعامل يأخذ حدين و يعيد قيمة واحدة.

ويوجد أيضاً معاملات أحادية unary operators تأخذ قيمة واحدة و تعيد قيمة أخرى فمثلا المعامل not يأخذ حداً إما false أو true ويعيد القيمة المعاكسة .

البرنامج يوضح لنا استخدام المعامل not:

```
Code
program test;
var res,sum:integer;
begin
res:=28;
if not((res=28)and(res mod 2=0))then
writeln('success')
else
writeln('fail');
if not (true) and(res=10) then
writeln('arabic syria');
readln;
end.
```

- في التعليمة not((res=28)and(res mod 2=0)) فإن المعامل not يأخذ حد وحيد وهو : (res=28)and(res mod 2=0) و قيمتة true ويكون لدينا not(true) و بالتالي يكون لدينا if false then
- في التعليمة if not (true) and(res=10) then فإن المعامل not يأخذ حد وحيد و هو (true) وبالتالي يكون: if false and true = false

خرج البرنامج السابق هو:

```
fail
```

معامل الإلحاق assignment operator:
يأخذ معامل الإلحاق =: حداً واحداً متحولاً كان أو قيمة و يلحق قيمة هذا الحد بمتحول و العنصر اليساري في عبارة الإلحاق يجب أن يكون متحولا حصراً و بشكل عام يجب أن يكون المتحول و الحد الموجود يمين متحول الإسناد من نفس نوع المعطيات . و فيما يلي بعض الأمثلة على عبارات الإلحاق:

Myinteger:=399;
Variable_integer:=my_integer;
Mystring:='hello to Syria arabic';
Myvalue:=55*yourvalue+2 /theirvalue;

معاملات المقارنة comparison operators:
نحتاج في البرامج التي نكتبها أحياناً إلى مقارنة قيمتين فمثلاً نريد مقارنة القيمة val1 و القيمة val2 لمعرفة أي القيمتين أقل و مثل هذه المقارنات يمكن أن تتطبق على المعلومات العددية و غير العددية و يكون ناتج المقارنة دائماً هو true أو false و تملك لغة باسكال ستة معاملات مقارنة (comparison operators) تدعى بالمعاملات العلائقية relational operators
و كل واحد من هذه المعاملات يأخذ حدين و يعيد إحدى قيمتين true أو false و معاملات المقارنة تنتمي تتعامل مع حدود عددية و غير عددية و هذه الحدود يجب أن تنتمي إلى نفس نوع المعطيات من أجل المقارنة.
مثال :لدينا تعابير مقارنة مقبولة:

'hello'='good bye';
1>=102

مثال: تعبير غير مقبول :
(* مقارنة مرفوضة لعدم تجانس بين أنواع المعطيات المقارنة*) ;'hello'=>1

معاملات المقارنة هي:

1. معامل المساواة = equality operator:يعيد هذا المعامل القيمة true إذا كان الحدان المقارنان متساويان و إلا فإنه يعيد false. مثال:

   'hello'='hello' (* true *)
   39=103 (* false*)

2. معامل عدم المساواة <> inequality operator:يعيد هذا المعامل القيمة true إذا كان الحدان المقارنان غير متساويان و إلا فإنه يعيد القيمة false. مثال:

   'hello'<>'hello' (* false *)
   39<>103 (* true*)

3. معامل القيمة الأكبر > greater than operator:يعيد هذا المعامل القيمة true إذا كان الحد اليساري أكبر من الحد اليميني و إلا فإنه يعيد القيمة false. مثال:

   'zebra'>'aardvak' (* true*)
   'arabic'>'syria' (* false*)
   39.99>236.1 (*false*)

4. معامل القيمة الأكبر أو المساواة => greater than or equal to operator:يعيد هذا المعامل القيمة true إذا كان الحد اليساري أكبر من أو يساوي الحد اليميني و إلا فإنه يعيد القيمة false. مثال:

   'syria'>='arabic' (*true*)
   12>=11 (*true*)

5. معامل القيمة الأقل < less than operator: يعيد هذا المعامل القيمة true إذا كان الحد اليساري أقل من الحد اليميني و إلا فإنه يعيد القيمة false. مثال:

   'good bye'<'university' (*true*)
   123<1036 (* true*)

6. معامل القيمة الأقل أو المساواة => less than or equal to operator:يعيد هذا المعامل القيمة true إذا كان الحد اليساري أقل أو يساوي الحد اليميني و إلا فإنه يعيد القيمة false. مثال:

```
123<=123        (*true)
1036<=3         (*false*)
'page'<='book'  (*false');
```

ملاحظة:معاملات المقارنة تعيد دائماً قيمة بوليانية true أو false.

أسبقية المعاملات operator precedence:

عندما نستخدم أكثر من معامل واحد في عبارة ما فإن أسبقية المعاملات تقرر ترتيب المعاملات الواجب تطبيقها أولاً و نحن نعلم في علم الجبر أن عملية الضرب تملك أسبقية على عملية الجمع إذ يمكن رؤية قانون الأسبقية هذا في التعبير الرياضي التالي 3*6+8 و الذي ناتجه هو 26 و ليس 42.

حيث في التعبير السابق يتم حساب 18=3*6 أولاً و ثم يُجمع الناتج مع 8 و هكذا كانت نتيجة التعبير السابق 26 .

و حتى نجعل ناتج التعبير السابق 42 يجب علينا إضافة أقواس لنتجاهل قانون أسبقية الضرب و بالتالي التعبير التالي 3*(6+8) ناتجه هو 42.

و بالإضافة إلى قانون الأسبقية هذا فإن المعاملات تُحسب دائماً وفق اتجاه ثابت إذ يُحسب التعبير التالي من اليسار إلى اليمين في حال كل المعاملات تملك نفس الأسبقية:

```
1 + 2 + 6 + 23 + 85 + 189
    3 + 6 + 23 + 85 + 189
        9 + 23 + 85 + 189
            32 + 85 + 189
                117 + 189
                      306
```

معامل الإلحاق يملك أقل أسبقية بالمقارنة مع كل معاملات لغة باسكال . وقانون الأسبقية يتأكد من أن قيمة التعبير الكلي الواقع على يمين عبارة الإلحاق قد حُسب قبل أن يُلحق هذه القيمة بالمتحول الواقع على يساره أما معاملات المقارنة فهي تملك نفس الأسبقية فهي تملك فيما بينها و هذه الأسبقية بالطبع أعلى من أسبقية معامل الإلحاق.

تعريف الثوابت constant definitions:

قد نحتاج في برامج التي نكتبها إلى استخدام قيماً ثابتة فعلى سبيل المثال افترض=تخيل أنك قمت بكتابة برنامج فيزيائي يصف حركة سقوط الأجسام عندئذ سوف تحتاج إلى قيمة تسارع الجاذبية الأرضيةو التي تساوي تقريباً 9.8 متراً في الثانية المربعة و هذه القيمة سوف تبقى نفسها أينما استخدمت ضمن البرنامج و لذلك يمكن اعتبار هذه القيمة ثابتة constant.

تسمح لغة باسكال أن نعين أسماً لمثل هذه القيم عندئذ يمكننا أن نشير إلى هذه القيم بواسطة أسمائها و تعريف الثوابت يتم وفق الصيغة الكتابية لتعريف الثوابت هي:

Const <اسم الثابت>=<قيمة>;

مثال يبين لنا تعريف عدة ثوابت بعد الكلمة المحجوز لأجل تعريف الثابت و هي const:

```
Const   value=10;  mystring='syria arabic';
        valueR=263.65;
```

نلاحظ أن تعريف الثابت هو إجراء مساواة بين اسم الثابت و القيمة المسندة إلى هذا الثابت حيث عند ترجمة البرنامج فإن المترجم يستبدل القيم المخصصة للثوابت (قيمة الثابت) عند كل ظهور لأسماء هذه الثوابت و السؤال هنا إذا كان المترجم يقوم باستبدال اسم الثابت بقيمته فما هي الفائدة من تعريف الثابت في البرنامج ؟؟؟؟ !!!!! .

الأسباب هي:

أولاً: يغدو البرنامج أكثر وضوحاً و أسهل فهماً فمثلا إذا رأى القارئ شيئاً شبيهاً بهذا الثابت acceleration Constant ومعناه"ثابت التسارع" بدلاً من أن يُفاجأ بقيمة مثل 9.8 ضمن البرنامج.

ثانياً: استخدام عملية تعريف الثوابت تمكننا من تغيير قيمة هذا الثابت بسهولة عند القيام بأي تعديل مستقبلي مطلوب ضمن البرنامج ، افترض = تخيل أنك استخدمت قيمة ثابت التسارع في مئة موقع مختلفاً ضمن البرنامج الفيزيائي عندئذ يمكنك كتابة القيمة 9.8 في كل هذه المواقع بدلاً من اسم الثابت و لكن افترض= تخيل أنك تريد تعديل هذا البرنامج لكي ينجز حساباته حول سقوط الأجسام على سطح القمر حيث يكون تسارع الجاذبية هناك هو 1.6 متراً في الثانية المربعة و لكي تعدل هذا البرنامج يجب استبدال القيمة 9.8 بالقيمة 1.6 في كل مكان وُجدت فيه ضمن البرنامج و قد تكون عملية البحث العام

search عن القيمة 9.8 و الاستبدال التلقائي replace لها بالقيمة 1.6 غير مجدي فقد يكون قد استخدمت القيمة 9.8 في حسابات أخرى غير الحسابات المتعلقة بتسارع الجاذبية ولذلك يجب فحص كل قيمة 9.8 موجودة ضمن البرنامج من حيث ضرورة استبدالها بالقيمة 1.6 أم لا و هذه الطريقة سوف تكون مملة و غير مضمونة و الفرصة كبيرة لنسيان قيمة يجب استبدالها بالإضافة إلى الوقت الضائع في عملية التعديل .
أما إذاً عرفت ثابتاً يمثل تسارع الجاذبية يصبح بإمكانك تعديل البرنامج بتغيير قيمة هذا الثابت ضمن تعريف الثوابت و عندئذ يقوم المترجم بعمل التغيير اللازم في البرنامج عوضاً عنك.

الإجراءات المستخدمة في عملية الإخراج output procedure:

تزودنا لغة باسكال بالإجراءين write و writeln لكتابة المعلومات على الشاشة أو على ملف حيث يقوم الإجراء writeln بإظهار المعلومات المراد إظهارها و ينتقل بعدها إلى بداية سطر جديد أما الإجراء write فهو يُظهر المعلومات المحددة و لكن لا ينتقل إلى سطر جديد و يبقى المؤشر عند نهاية أخر رمز كتبه.
و الفرق الوحيد بين الإجراءين write و writeln أن الإجراء writeln بعد ان يُظهر المعلومات المحددة يُرسل اتحاداً بين رمزي إرجاع الحاملة carriage return ورمز التغذية السطرية line feed و يرمز لهذا الاتحاد بالرمز CRLF أما الإجراء write فلا يُرسل أي رمز بعد أن يُظهر جميع المعلومات المخصصة له و البرنامج التالي يوضح لنا عملية استدعائي هذين الإجراءين :

Code
```
program test;
var testvalue:integer;
begin
testvalue:=22333;
write('Just a sentence, without CRLF.');
writeln('this continues on the same line.');
writeln;
writeln;
writeln('there should be two blank lines abov this one.');
writeln;
write('Eight = ',8,'; nine = ',9);
writeln('; ten = ',10);
writeln(  'testval =',testvalue);
readln;
end.
```

```
Just a sentence, without CRLF.this continues on the same line.


there should be two blank lines abov this one.

Eight = 8; nine = 9; ten = 10
testval =22333
```

الإجراءين write و writeln يمكن أن يأخذا عدد متغيراً (غير محدد) من المتحولات الوسيطية arguments و إذا قمنا باستدعاء الإجراء writeln بدون متحولات وسيطية عندئذ يرسل الأمر CRLF فقط. أي ينقل المؤشر إلى بداية سطر جديد إما إذا استدعينا الإجراء write بدون متحولات وسيطية فإن هذا الإجراء **لا يقوم بأي** عمل مع العلم أن هذا الاستدعاء صحيح تركيبياً.
و هذه الإجراءات تتعامل مع أنواع مختلفة من المعطيات. وعندما نمرر أكثر من متحول وسيطي إلى الإجراءين writeln و write يجب أن نفصل بين المتحولات بفاصلة (,) و يمكننا ترك مسافات فارغة بين المتحولات كما نريد و المثال التالي يبين لنا استدعائين متكافئيين و مقبولين :

```
Write(agr1,agr2,arg3);
Write( arg1  ,   arg2,   arg3);
```

ويمكن أن تتجاوز لائحة المتحولات الوسيطية السطر الواحد و لكن يجب الانتباه إلى وقوع مكان الانتقال إلى السطر التالي بين متحولين و ليس في منتصف المتحول الوسيطي الواحد فعلى سبيل المثال :

لدينا الاستدعاءات الثلاثة التالية: حيث الاستدعاء الأول صحيح و الاستدعء الثاني غير صحيح و الاستدعاء الثلث يعطينا حلاً لمشكلة توزيع المتحولات الوسيطية حيث قسم السلسلة الرمزية إلى سلسلتين ووضعهما على سطرين متتاليين:

```
First call(true) : writeln(Arg1,arg2,' A string argument ',
                        Arg4,
                        'another string argument',arg6);
Second  call(false): writeln(arg1, arg2, ' A string argument
                        broken over tow lines',
                        aarg4,
                        'another string argument',arg6);
Third call(true): writeln(arg1.arg2,'A string argument',
                  'broken over tow lines',
                  arg4,
                  'another string argument ',arg6);
```

تستخدم عادة الفراغات بين المتحولات أو تتخدم عملية توزيع المتحولات على أسطر من أجل تسهيل عملية قراءة البرنامج و فهمه.

يقوم الإجراءن write و writeln بالكتابة على ملف إذا كان أول متحول في لائحة المتحولات الوسيطية لهذين الإجراءين هو اسم ملف إما إذا كان اول متحول وسيطي ليس اسم ملف فإن الكتابة تتم على جهاز الخرج القياسي (الشاشة).

الإجراءات المستخدمة في عملية الإدخال input procedure:

تزودنا لغة باسكال بالإجراءين read و readln لقراءة المعلومات من ملف أو من لوحة المفاتيح keyboard و يمكن لهذين الإجراءين أن يأخذا عدداً متغيراً(غير محدد) من المتحولات الوسيطية حيث لدينا مجموعة من الاستدعاءات لروتينات الإدخال:

```
Readln(filearg,myint,yourint);
Read(filearg,myint,yourint);
Readln(val1);
```

الاستدعاء الأول من الاستدعاءات السابقة يقرأ قيمتين من الملف الذي حُدد اسمه بواسطة المكتحول الوسيطي الأول filearg مفترضاً = متخيلاً أن هذا الملف قد صُرح عنه مسبقاً و يجب أن تكون القيمة الأولى المقروءة من هذا الملف مخصصة للمتحول myint و القيمة الثانية المقروءة مخصصة للمتحول yourint و بعد أن تتم عملية قراءة هاتين القيمتين ينتقل الإجراء إلى بداية السطر التالي ضمن الملف.

الاستدعاء الثاني للإجراء read له نفس وظيفة الاستدعاء الأول إلا إنه لا يقوم بالانتقال إلى بداية السطر التالي بعدد قراءة القيمتين المحددتين.

افترض=تخيل أن المعلومات المخزنة في ملفك هي كما يلي:

f.txt		
39	27	293
54		

و السؤال ما هو ناتج استدعاء الإجراءات التالية (ما هي قيمة المتحول theirint):

```
Readln(filearg,myint,yourint);
Readln(filearg,theirint);
```

بعد قراءة القيمة 39 و إلحاقها بالمتحول myint و قراءة القيمة 27 و إلحاقها بالمتحول yourint ضمن الاستدعاء الأول للإجراء readln ينتقل عندها إلى بداية سطر جديد في الملف f.txt يتم تجاهل القيمة 293 و هكذا فإن القيمة 54 تقرأ بواسطة الاستدعاء الثاني للإجراء readln و تُلحق بالمتحول theirint .

سؤال لنعدل استدعاء الإجراءين السابقين إلى الاستدعئين التاليين:

```
Read(filearg,myint,yourint);
Read(filearg,theirint);
```

في هذه الحالة تُلحق القيمتان 39 و 27 بالمتحولين myint و yourint كما في الاستدعاءات السابقة أما القيمة 293 فسوف تُعطى للمتحول theirint لأن الاستدعاء الأول لا ينتقل إلى بداية سطر جديد بعد قراءة القيمتين المحديتين.

أما إذا كان استدعاء الإجراءين read و readln لا يتضمن متحول ملف مكان المتحول الأول فإن الإجراءات السابقة تفترض=تتخيل أن المعلومات سوف تأتيها من جهاز الدخل القياسي(لوحة المفاتيح) و لذلك فإن كلا الاستدعائين التاليين ينتظر من المستخدم كتابة = إدخال ثلاثقيم :

```
Read(arg1,arg2,arg3);
Readln(arg1, arg2,arg3);
```

إذا كانت القيم المُراد إدخالها قيماً عددية فإننا نستطيع أن ندخل بعض أو كل القيم على سطر واحد طبعاً مع الفصل بين هذه القيم بفراغات أو يمكننا إدخال هذه القيم على أسطر مستقلة و عندئذ يجب أن يتم الضغط على مفتاح الإدخال enter بعد وضع آخر قيمة من كل سطر.

إذا استدعينا الإجراء readln بدون متحولات وسيطية فإن البرنامج سيتوقف عندما يصل إلى عبارة استدعاء الإجراء هذا و ينتظر حتى نضغط على مفتاح الإدخال enter ليكمل مهمته. و هذه طريقة جيدة لإجبار البرنامج على التوقف المؤقت حتى نقوم بتفحص خرج البرنامج

إذا طلبنا من كلا الإجراءين read و readln أن يقرأ قيماً من نوع معين و كتبنا قيماً من نوع مختلف عندئذ سينتج لدينا خطأ أثناء التنفيذ run-time error و يتوقف البرنامج مع خطأ التفيذ (crash) فمثلا إذا أخبرت الإجراء read بأن يقرأ عدداً ووجد عوضاً عنه سلسلة رمزية أو حرف فسوف نحصل على خطأ في التنفيذ و ليس أثناء الترجمة.

بفرض أن محتويات الملف f.txt:

f.txt
39 27

و المطلوب ما هو خرج البرنامج التالي:

Code

```
program test;
var f:text;
a,b,c:integer;
begin
assign(f,'c:\f.txt');
reset(f);
a:=02;
b:=06;
c:=09;
readln(f,a,b,c);
writeln('a = ',a,';b = ',b
,
';c = ',c);
close(f);
readln
end.
```

الخرج هو:

A. **a = 39;b = 27;c = 0**
B. a = 39;b = 27;c = 9
C. a = 39;b = 27;c = 09
D. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه
E. None of the above

الأعداد الحقيقية Real Numbers:

نستطيع في بعض الأحيان إنجاز أعمالنا باستخدام الأعداد الصحيحة فقط whole numbers و في بعض الأحيان نحتاج إلى أعداد تحتوي على أجزاء كسرية مثل 1.5 و 39.33 .

و هذه الأعداد تدعى اعداد حقيقية real numbers و الأعداد الحقيقية هذه تستخدم النقطة . لتفصل بين القسم أو الجزء الصحيح و القسم الكسري للعدد و تدعى عادة بالفاصلة العشرية لأنها تُستخدم مع الأعداد ذات الأساس العشري و لكن من الممكن أن تستخدم في تمثيل أعداد ذات أساس يختلف عن 10 فمعظم مترجمات اللغات تمثل القيم بشكلها الثنائي أي باستخدام الأساس 2.

نوع المعطيات عدد حقيقي pascal 's real types:

تملك لغة باسكال نوعاً مسبق التعريف هو نوع معطيات عدد حقيقي real لتمثيل الأعداد الحقيقية إذ يُخصص مترجم لغة باسكال ستة بايتات من الذاكرة لكل متحول حقيقي ومن خلال سعة التخزين هذه يكون مجال الأعداد الممكن تخزينها هي:

من أجل الأعداد الموجبة $2.9 * 10^{-39}$ حتى $1.7 * 10^{38}$ و

من أجل الأعداد السالبة $-2.9 * 10^{-39}$ حتى $-1.7 * 10^{38}$ و

يوضح البرنامج التالي كيفية التصريح عن المتحولات من النوع الحقيقي و كيفية استخدام هذه المتحولات :

Code

```
program test;
var rvalue:real;
procedure getreal(message:string; var value:real);
begin
write(message,'?');
readln(value);
end;
begin {main program }
getreal('value',rvalue);
writeln(rvalue);    (* use default output format *)
writeln(rvalue:10:5);    (* use specified output format *)
readln;
end.
```

يحصل البرنامج السابق على قيمة من المستخدم و يُظهر هذه القيمة وفق شكلي إخراج و شكل إخراج مثل هذه القيم سوف يشرح لاحقاً.

```
value?1267
 1.2670000000E+03
1267.00000
```

المهندس خالد ياسين الشيخ

خرج البرنامج السابق شبيه بالتالي:

الصيغة الكتابية للأعداد الحقيقية syntax for real values:

تملك الأعداد الحقيقية شكلاً خاصاً في لغة باسكال إذ يمكن أن تبدأ هذه الأعداد بإحدى الإشارتين + أو - فعلى سبيل المثال الأعداد التالية هي قيم مقبولة في لغة باسكال :

-353.5 , +35.36 , 353.5

يجب ان تحتوي الأعداد الحقيقية على الأقل رقماً واحداً و يمكن أن تحتوي على فاصلة عشرية واحدة فقط و إذا تضمن العدد فاصلة عشرية يجب أن يحتوي العدد على رقم واحد على الأقل قبل (على يسارها)و يمكن أن يكون هذا الرقم صفراً إذا تطلب الأمر ذلك.

و كمثال على أعداد حقيقية مقبولة في لغة باسكال نورد الأعداد التالية:

35.47 , +5.5 , 499999 , 0.1732 , -69.

و أخرى غير مقبولة:

+.6 , 5.+6 , 6.5.3 , -.9

يمكن أن نستخدم التدوين العلمي scientific notation لكتابة الأعدد الحقيقية و التدوين العلمي يُعبر عنه على شكل قيمة و أس إذ يحدد الأس exponent قوة العدد 10 الواجب رفع العدد 10 إليها قبل ضربه بالقيمة value للحصول على العدد النهائي فمثلاً الأعداد التالية كلها طرق مقبولة و مختلفى لتمثيل العدد 15 بالتدوين العلمي في لغة باسكال:

0.15E2 1.5E1 15e0 1500E-2

الحرف e أو E يُحدد بداية الأس و العدد الذي يأتي بعد أحد هذين الحرفين يمثل قوة الأساس 10 الذي يجب ضربه بالقيمة لنحصل على العد الكلي .

كل قوة موجبة positive power للعدد 10 تعني ضرباً آخر للعدد بالقيمة 10 أي أن 0.15E2 تعني $(0.15 * 10^2)$ أو (0.15*10*10) أو (0.15*100) .

و بشكل مشابه فإن القوة السالبة negative power للعدد 10 تعني تقسيماً جديداً للعدد على 10 أي 1500E-2 يعني (1500/10/10) أو (1500/100).

و يجب الانتباه إلى ضرورة أن يُتبع الحرف e أو E برمز الإشارة (+ أو -) و أن يكون الأس دائماً عدداً صحيحاً و يجب أن نتذكر دائماً أن شكل الأعداد الحقيقية المسموح بها في لغة باسكال تأتي على الشكل التالي:

1. رمز إشارة اختياري للعدد (+ أو -).
2. رقم أو أكثر مع إمكانية أن يكون هذا الرقم صفراً.
3. فاصلة عشرية اختيارية متبوعة برقم أو أكثر و يمكن أن لا تُتبع بأي رقم.
4. رمز أس اختياري (E أو e) متبوعاً برمز إشارة اختياري (+ أو -) متبوعاً بعدد صحيح.

إظهار الأعداد الحقيقية displaying real values:

عند إظهار الأعداد الحقيقية يمكننا أن نتحكم بشكل خرج هذه الأعداد أو أن نترك النظام يستخدم الشكل القياسي في عملية الإظهار هذه. ففي الاستدعاء الأول للإجراء writeln ضمن البرنامج السابق استخدمنا شكل الإظهار النظامي للأعدد الحقيقية حيث تكتب لغة التربو باسكال الأعداد الحقيقية وفق التدوين العلمي في الحالة العامة.

أما الاستدعاء الثاني للإجراء writeln ضمن البرنامج السابق يُظهر القيمة rvalue وفق طريق معينة في عملية الإظهار تلك حيث تحدد الرموز 10:5 الموجودة بعد اسم المتحول في عبارة الاستدعاء تلك عرض المجال ودقة الخرج حيث:

1. يحدد عرض المجال field width عدد الأعمدة الواجب استخدامها في إظهار العدد و عرض المجال في المثال السابق هو 10 أعمدة.
2. تحدد الدقة precision عدد الأرقام بعد الفاصلة العشرية (على يمينها) و في المثال السابق حددت الدقة على أنها خمسة أرقام بعد الفاصلة العشرية.

الفاصلة العشرية و الإشارة متضمنتان في عرض المجال لذلك إذا حددنا عرض المجال بعشرة أعمدة فإن الفاصلة العشرية و إشارة العدد تستخدمان عمودين من هذه الأعمدة و يبقى عندئذ للعدد ثماني أعمدة فقط و في حال كون العدد المراد كتابته أكبر من عرض المجال المخصص له فإن الإجراء writeln يتجاهل تعليمة الشكل المحددة له و يستخدم أعمدة تكفي لإظهار العدد كاملاً و إذا احتاج الإجراء إلى أعمدة أقل من الأعمدة المخصصة له فإن الفراغات سوف توضع على يسار هذا العدد أي أن الرقم اليميني في هذا العدد سوف يوضع في آخر عمود مخصص وفق عرض المجال لهذا العدد.

التقريب في الأعداد الحقيقية real values as approximations:

نحن نعلم في علم الرياضيات الذي لا حدود له أن الأعداد الحقيقية لا نهائية infinite numbers و في الحقيقة يمكننا إيجاد أعداد لا تحصى بين أي قيمتين حقيقيتين و لكن هناك عدداً منتهياً من البتات المستخدمة في تخزين الأعداد الحقيقية و هذا يعني عدم إمكانية تخزين أي قيمة حقيقية كانت ضمن متحول من النوع الحقيقي لأنه و كما ذكرت يوجد عدد محدد من البتات لتمثيل الأعداد الحقيقية و القيم الممكن تخزينها فعلياً هي قيم مقربة و ينشأ هذا التقريب من عدم وجود بتات كافية لتمثيل أرقام العدد المطلوب تخزينه فعلى سبيل المثال يُمكننا تمثيل أعداد حقيقية في لغة التربو باسكال بطول 11 رقماً على الأكثر و هنا ينشأ الخطأ لأن عملية التقريب أو التدوير للعدد تحدث تلقائياً أثناء الحسابات . فمثلا افترض=تخيل أننا نود جمع العددين التاليين:

1.12345678901234567890 و 0.12345678901234567890

نلاحظ أن القيمتين السابقتين قد تجاوزتا حدود الدقة للأعداد الحقيقية و هي 11 رقماً و لذلك يُنجز النظام تلقائياً حسابات ذات أرقام أقل و سوف يتأثر الناتج حتماً بهذه الحسابات المقربة و هذا يعني أن الحسابات المطبقة على الأعداد الحقيقية سوف تُعطي نتائج صحيحة تقريباً و خاصة بالنسبة للحسابات المطبقة على أعداد ذات مجالات كبيرة او على أعداد تتطلب مجال دقة كبير جداً أو على قيم تتطلب حسابات كثيرة.

ومن حسن الحظ أن مثل هذه الأخطاء تكون من المراتب الصغيرة جداً و تظهر عادة عندما تكون القيمة المطلوب حسابها صغيرة جداً و تحتاج إلى خانات دقة أكبر من مجال الدقة المحدد لهذه الحسابات فمثلا قلما نحتاج إلى نتائج ذات دقة تتجاوز خمسة أو ستة مواضع بعد الفاصلة العشرية و لكن الأخطاء تحدث عادة في الموضع التاسع أو العاشر بعد الفصلة .

لاحظ البرنامج التالي:

Code
program test; var v1,v2:real; begin v1:=1.1294567891832; v2:=1.1234567891232; writeln(v1);

```
writeln(v2);
writeln(v1:6:2);
writeln(v2:6:2);
readln;
end.
```

```
1.1294567892E+00
1.1234567891E+00
  1.13
  1.12
```

خرج البرنامج السابق هو:

المعاملات المطبقة على القيم الحقيقية operators for real values:
يحدد التصريح عن المتحولات كمية الذاكرة المستخدمة لكل متحول و يقرر بالإضافة إلى ذلك أي المعاملات يمكن تطبيقها على هذا النوع من المتحولات إذ يمكننا جمع و ضرب عددين و لكن لا نستطيع أن نجمع أة أن نضرب رمزين.

معامل الإلحاق و معامل المقارنة assignment and comparison operators:
يمكننا استخدام معامل الإلحاق و معاملات المقارنة مع المتحولات و القيم الحقيقية و البرنامج التالي يوضح ذلك:

Code

```
program test;
var val1,val2:real;
const cutoff=1000.0;
    twicecutoff=2000.0;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin {main program}
getreal('value:',val1);
if val1>cutoff then
val2:=val1
else
val2:=cutoff;
writeln('val1 = ',val1:10:3,
        '   val2 = ',val2:10:3);
if(val1<=Twicecutoff) then
val2:=Twicecutoff;

writeln('val1 = ',val1:10:3,
        '   val2 = ',val2:10:3);
readln;
end.
```

يستخدم البرنامج السابق معاملي مقارنة ضمن عبارة if و ثلاث معاملات إلحاق كلها استخدمت في تغيير قيمة المتحول val2
ومن أجل قيم للدخل هي 124.36 و 1536.3 و 15263 يكون خرج البرنامج السابق هو:

```
value: 124.36
val1 =     124.360     val2 =    1000.000
val1 =     124.360     val2 =    2000.000

value: 1536.3
val1 =    1536.300     val2 =    1536.300
val1 =    1536.300     val2 =    2000.000

value: 15263
val1 =   15263.000     val2 =   15263.000
val1 =   15263.000     val2 =   15263.000
```

و من الجدير ذكره هو خطر المقارنة بين قيمتين حقيقتين كما في البرنامج التالي :

Code

```
program test;
const increment=0.003;
var val:real;
begin
val:=0.9;
while(val<>1.2)do
begin
writeln('val = ',val:10:7);
val:=val+increment;
if(val>1.2)then
readln;
end;
writeln('val = ',val:10:7);
readln;
end.
```

يدخل البرنامج السابق في حلقة لا نهائية حيث يصل البرنامج ظاهرياً إلى القيمة 1.2 و التي توقف حلقة while و لكن استمرار تنفيذ الحلقة while يُشير إلى أن القيمتين 1.2 و قيمة المتحول val ليستا متطابقتين تماماً.
فعندما نبحث عن مساواة أو عدم مساواة بين قيمتين حقيقتين تقوم باسكال بمقارنة نماذج البتات bit pattern لكلتا القيمتين و في البرنامج السابق قيمة المتحول = المتغير val تنتج من خلال حسابات متكررة ينتج عنها بعض الأخطاء الحسابية مما يسبب اختلافاً في نماذج بتات هذا هذ المتحول عن نماذج بتات العدد 1.2.

المعاملات الحسابية arithmetic operators:
يمكننا استخدام المعاملات الحسابية العادية على القيم الحقيقية مثل الجمع و الطرح و الضرب و القسمة فمعامل الجمع + و معامل الطرح – يقومان بعملية على قيمتين حقيقتين لإنتاج قيمة حقيقية ثالثة و البرنامج التالي يبين لنا جمع و طرح قيمتين حقيقتين :

Code

```
program test;
```

```
const emptystring='';
plusop='+';
var val1,val2:real;
operation:string;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
getstring('+ or - (enter to stop) ',operation);
while( operation <>'+') and (operation <>'-') and (operation<>'')do
getstring('+ or - (enter to stop) ',operation);
while(operation <> emptystring)do
begin
getreal('value 1: ',val1);
getreal('value 2: ',val2);
if(operation=plusop)then
writeln(val1+val2:10:3)
else
writeln(val1 - val2:10:3);
getstring('+ or - (enter to stop) ',operation);
while( operation <>'+') and (operation <>'-') and (operation<>'')do
getstring('+ or - (enter to stop) ',operation);
end;
writeln('press enter to leave program.');
readln;
end.
```

لاحظ المتحولات الوسيطية الموجودة ضمن عبارة استدعاء الإجراء writeln في البرنامج الرئيسي السابق فعلى الرغم من طول الحدود ضمن الأقواس إلا أن هذا الاستدعاء يملك متحولاً وسيطياً واحداً إذ تستبدل التعابير التالية بقيمة حقيقية قبل القيام بعملية الاستدعاء :

Val1+val2

Val1-val2

و لا يبقى ضمن أقواس الإجراء عند استدعائه إلا القيمة المخصصة له.

و لاحظ أن الثابت emptystring قد عُرف على أنه ثابت من النوع سلسلة رمزية و لكن بدون رموز كما يحدث عندما نضغط مفتاح الإدخال enter بدون أن نكتب أي رمز قبل ضغطه.

خرج البرنامج السابق شبيه بالتالي:

المهندس خالد ياسين الشيخ

المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com
Syrian Arab Republic
دمشق-معضمية الشام

```
+ or - (enter to stop)  +
value 1:  125.00
value 2:  123.00
   248.000
+ or - (enter to stop)  -
value 1:   12.36
value 2:    1.30
    11.060
+ or - (enter to stop)  +
value 1:  100.00
value 2:    0.50
   100.500
+ or - (enter to stop)
press enter to leave program.
```

معامل الضرب * و معامل القسمة / يعملان في لغة باسكال كما في علم الرياضيات الذي لا حدود له حيث أن القسمة على صفر غير معرف في علم الجبر و في لغة باسكال تسبب محاولة قسمة عدد على صفر خطأ في التنفيذ و توقفاً للبرنامج والبرنامج التالي يبين لنا عملية ضرب و قسمة عددين حقيقيين :

Code

```
program test;
const emptystring='';
timesop='*';
var val1,val2:real;
operation:string;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
begin
getstring('* or / (enter to stop) ',operation);
while( operation <>'*') and (operation <>'/') and (operation<>'')do
getstring('* or / (enter to stop) ',operation);
while(operation <> emptystring)do
begin
getreal('value 1: ',val1);
getreal('value 2: ',val2);
if(operation=timesop)then
writeln(val1*val2:10:3)
else
writeln(val1 / val2:10:3);
getstring('* or / (enter to stop) ',operation);
while( operation <>'*') and (operation <>'/') and (operation<>'')do
getstring('* or / (enter to stop) ',operation);
end;
writeln('press enter to leave program.');
readln;
```

```
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
* or / (enter to stop)  *
value 1:  100.00
value 2:     2.00
   200.000
* or / (enter to stop)  /
value 1:  250.00
value 2:    50.00
     5.000
* or / (enter to stop)  *
value 1:   25.00
value 2:     5.00
   125.000
* or / (enter to stop)  /
value 1:  100.00
value 2:   20.00
     5.000
* or / (enter to stop)
press enter to leave program.
```

البرنامج السابق متطابق تماماً مع البرنامج الذي سبقه مع وجود الاستثناءات التالية:

1. تم استبدال الثابت plusop بالثابت timesop.
2. المعاملات ضمن رسائل الاستدعائين للإجراء getstring قد تغيرت من + إلى * ومن – إلى / .
3. تغيرت المعاملات أيضاً في كلا الاستدعاءين للإجراء writeln ضمن البرنامج الرئيسي.

أسبقية المعاملات المطبقة على الأعداد الحقيقية precedence of operators for reals :

المعاملات الحسابية تقع عادة ضمن مستويين من مستويات الأسبقية فمعاملي الضرب و القسمة يملكان أسبقية واحدة و مستوى هذه الأسبقية أعلى من مستوى الأسبقية الذي يقع فيه كل من معاملي الجمع و الطرح و أسبقية كل المعاملات الحسابية أعلى من أسبقية معاملات المقارنة و كذلك أعلى من أسبقية معامل الإلحاق.

ملاحظة:

- إذا كتبنا رموزاً غير رقمية خطأً بينما البرنامج يتطلب منا قيماً حقيقية فإن البرنامج سيتوقف و ينتج لدينا خطأً في التنفيذ.
- تمكننا لغة التربو باسكال من تمثيل قيم حقيقية ذات مجالات كبيرة و مع ذلك قد نتجاوز هذه الحدود و يحدث ما يُعرف بالفيض overflow فإذا حصل لدينا زائد في قيمة عدد حقيقي يتوقف البرنامج و نحصل على رسالة خطأ في التنفيذ.

أنواع المعطيات الأعداد الصحيحة whole numbers types:

تزودنا لغة التربو باسكال بأنواع معطيات متعددة تُستخدم في تخزين الأعداد الصحيحة و تختلف هذه الأنواع في حجم مواضع التخزين المخصصة لكل نوع و في إمكانية تمثيل الأعداد السالبة في هذه الأنواع أم لا.

نوع المعطيات عدد صحيح the integer type:

يُستخدم هذا النوع لتمثيل الأعداد الصحيحة من -32768 إلى 32767 إذ يُخصص المترجم بايتين (حجرتين من حجرات الذاكرة أي 16 بتاً) لكل قيمة من النوع صحيح integer ومن خلال هذا التمثيل يُمكننا تخزين قيم ضمن المجال من -2^{15} و حتى $2^{15}-1$ حيث يُمثل البت السادس عشر (البت الموجود في أقصى اليسار) إشارة القيمة فإذا كان هذا البت يساوي 1 تكون القيمة سالبة و إذا كانت قمة هذا البت يساوي 0 تكون القيمة موجبة.

تملك لغة التربو باسكال ثابت مسبق التعريف يدعى maxint قيمته تساوي أعظم قيمة صحيحة يمكن استخدامها و قيمة هذا الثابت تساوي 32767 و البرنامج التالي يوضح كيفية التصريح عن المتحولات و كيفية تسميتها وإظهارها:

Code

```
program test;
var int1,int2:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,'? ');
readln(value);
end;
begin
getinteger('value 1',int1);
getinteger('value 2',int2);
writeln('the sum of ',int1,' and ',int2,
      ' is ',int1+int2);
writeln('maxint = ',maxint);
readln;
end.
```

خرج البرنامج السابق من أجل int2 =12 and int2=13 هو:

```
value 1? 12
value 2? 13
the sum of 12 and 13 is 25
maxint = 32767
```

صرحنا في البرنامج السابق عن متحولين من النوع الصحيح و قام المترجم بحجز أربعة بايتات من الذاكرة من أجل هذين المتحولين.

لا يمكننا بلغة التربو باسكال استخدام الفاصلة بين أرقام العدد الصحيح و لكن نستطيع فقط أن نستخدم رمز الإشارة + أو – و بشكل اختياري في بداية العدد الصحيح.

فعلى سبيل المثلل:

الأعداد التالية صحيحة و مقبولة:

+8213 -2365

أما الاعداد التالية فهي غير مقبولة:

23.36 +28,455 -32,500

التحكم بشكل الخرج controlling output format:

تستخدم لغة التربو باسكال تلقائياً عدداً من الأعمدة بقدر ما تحتاج لكتابة العدد الصحيح و هذه التلقائية شيء مريح و لكنها تعطينا تسلسل نتائج غير متوقعة فعلى سبيل المثال لدينا البرنامج التالي:

Code

```
program test;
var val1,val2,val3:integer;
procedure getinteger(message:string; var value:integer);
begin
write(message,'? ');
readln(value);
end;
begin
getinteger('value 1',val1);
getinteger('value 2',val2);
getinteger('value 3',val3);
writeln(val1,val2,val3);
```

```
readln;
end.
```

خرج البرنامج السابق من اجل val3=26 and val2=25 and val1=12 هو كالتالي:

```
value 1? 12
value 2? 25
value 3? 26
122526
```

من أجل تجنب مثل الأخطاء الطباعية يمكننا وضع فراغات بين القيم مثل الاستدعاء التالي للإجراء:

Writeln(val1,'   ', val2,'   ',val3);

أو يمكن نحدد شكلاً للخرج بتحديد عدد الأعمدة الواجب استخدمها في كتابة القيمة فمثلاً العبارة التالية تكتب قيمة المتحول val وتحدد له حقلا عرضه 15 عمود.

Writeln(val:15);

إذا كانت القيمة المراد كتابتها أكبر من عرض الحقل المخصص لها فإن عرض الحقل هذا يتم تجاهله و تكتب القيمة كما هي إما إذا كانت القيمة المراد كتابتها أقصر من عرض الحقل المخصص لها فإن الجزء الأيسر من الخرج سيُملأ بالفراغات.ومثل هذه الطريقة في الإخراج و اليت يوضع فيها آخر رقم من القيمة في آخر عمود محدد بواسطة عرض الحقل تدعى بالتدفق اليميني flash right أو التحديد اليميني right justified.

المعاملات المطبقة على الأعداد الصحيحة operator for integer values:

بالإضافة إلى معامل الإلحاق و معاملات المقارنة و التي تطبق على كل نوع من أنواع المعطيات يمكننا استخدام المعاملات التالية على الحدود من النوع الصحيح:

المعاملات الحسابية arithmetic operators:

يمكننا أن نجمع أو نطرح أو نضرب أي قيمتين صحيحتين فمن أجل جمع أي قيمتين صحيحتين نستخدم معامل الجمع + و العبارات التالية تستخدم هذا المعامل:

```
Sum:=val1+val2;
Total:=sum+trunk(10.66);
```

وبشكل مشابه فإن معامل الطرح – يعطينا إمكانية طرح قيمة من قيمة أخرى كما هو موضح في العبارات التالية:

```
Sum:=val1- val2;
Total:=sum-trunk(10.66);
```

ومعامل الضرب * يعطينا قيمة ناتجة عن ضرب حدين ببعضهما ما تضح العبارات التالية:

```
product:=val1*val2;
Total:=sum*trunk(10.66);
```

أما عملية القسمة نعلم أن ناتج قسمة عددين صحيحين يعبر عنه بقيمة صحيحة و باقي فمثلاً:ناتج قسمة 17 على العدد 3 هو 5 و الباقي 2 و هذا يعني أن العدد 3 موجود في العدد 17 خمس مرات أي (3*5=15) و العدد 2 ناتج طرح العدد 15 من العدد 17 ويسمى بالباقي.

تملك لغة باسكال معاملين لإعطائنا ناتج قسمة عددين صحيحين و باقي قسمة هذين العددين فمعامل قسمة الأعداد الصحيحة هو div و هذا المعامل يأخذ حدين صحيحين مثل op1 و op2 و يعيد ناتج قسمة هذين العددين. حيث التعبير op1 div op2 يعطينا عدد المرات التي يتواجد فيها op2 ضمن op1 متجاهلاً أي باقي.

و البرنامج التالي يوضح المعامل div:

Code

```
program test;
const interrorvalue=-32768;
op='div';
var val1,val2,result,count:integer;
procedure getinteger(message:string; var value:integer);
```

```
begin
write(message,'? ');
readln(value);
end;
function intdiv(dividend,divisor:integer):integer;
begin
if divisor=0 then
intdiv:=interrorvalue
else
intdiv:=dividend div divisor
end;
begin
getinteger('value 1',val1);
getinteger('value 2',val2);
for count:=1 to 4 do
begin
result:=intdiv(val1,val2);
writeln(val1:3,' ',op,val2:2,' = ',result);
val1:=result;
end;
readln;
end.
```

خرج البرنامج السابق من أجل val1=260 and val2=8هو:

```
value 1? 260
value 2? 8
260 div 8 = 32
 32 div 8 = 4
  4 div 8 = 0
  0 div 8 = 0
```

المهندس خالد ياسين الشيخ

يستخدم المعامل mod لمعرفة باقي القسمة .لنفترض أن قيمة المتحول val هي 78 فأن قيمة التعبير التالي تساوي 9 :

Val mod 23;

حيث أن (78-69=9).
إن تطبيق المعامل mod سوف يتراوح بين القيمة 0 و قيمة أقل من قيمة المعامل الأيمن (الثاني) و هو هنا (23) بواحد وهكذا فإن ناتج التعبير السابق سوف يكون يملك قيمة تتراوح بين 0 و 22 و هذه القيمة تتعلق بقيمة الحد المقسوم و هو المتحول val.
و البرنامج التالي يُوضح لنا كيفية استخدام المعامل mod:

Code

```
program test;
const maxtrials=500;
var result,count,headcount,rowcount:integer;
begin
randomize;
headcount:=0;
rowcount:=0;
for count:=1 to maxtrials do
```

```
begin
result:=random(maxint);
if result mod 2=0 then
begin
write('.');
rowcount:=rowcount+1;
end
else
write(' ');
if count mod 50=0 then
begin
writeln(rowcount:3);
headcount:=headcount+rowcount;
rowcount:=0
end
end;
writeln(headcount,' heads');
readln;
end.
```

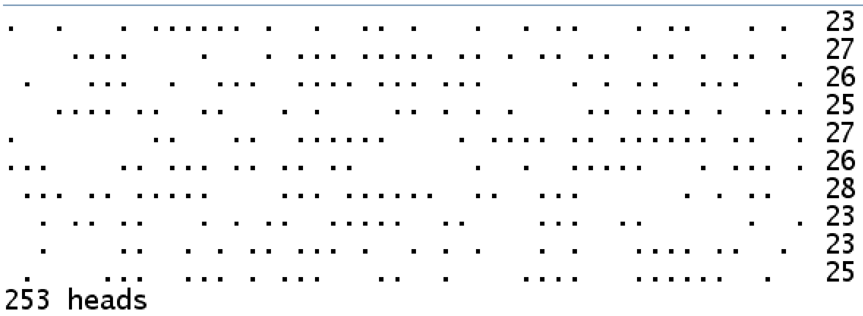
إن استدعاء التابع مسبق التعريف random مع الثابت maxint يولد قيم عشوائية صحيحة ضمن المجال ما بين 0 و maxint-1 أي ضمن المجال 0...32766 فإذا كانت القيمة التي يعيدها التابع random زوجية فإن البيرنامج يكتب نقطة و يُزيد العداد rowcount بمقدار واحد إما إذا كانت القيمة فردية فإن البرنامج يكتب فراغ و يكرر العمل حتى 50 محاولة و بعدها ينتقل إلى سطر جديد حيث يقوم بفحص الشرط count mod 50 =0 و خرج البرنامج السابق شبيه بالتالي:

Code

```
program test;
const maxtrials=500;
var result,count,headcount,rowcount:integer;
begin
randomize;
headcount:=0;
rowcount:=0;
for count:=1 to maxtrials do
begin
result:=random(maxint);
if result mod 2=0 then
begin
write('.');
rowcount:=rowcount+1;
end
else
write(' ');
if count mod 50=0 then
begin
writeln(rowcount:3);
headcount:=headcount+rowcount;
rowcount:=0
end
end;
writeln(headcount,' heads');
```

khaledyassinkh@gmail.com

```
readln;
end.
```



أسبقية المعاملات الحسابية precedence for atithmetic operators:

معاملات الجمع و الطرح و الضرب في الأعداد الصحيحة هي نفسها في الأعداد الحقيقية و لذلك فإن لها اسبقية مثل أسبقيتها بالنسبة للأعداد الحقيقية أما المعاملان Div و mod و للذان يُمثلان معاملان القسمة فلهما نفس أسبقية معامل الضرب و يقعان مع معامل الضرب في مستوى واحد.

القيم الست عشرية hexadecimal values:

نحن نستخدم في حياتنا اليومية أرقاماً عشرية تستخدم العدد 10 أساساً لها و في هذا النظام نتعامل مع عشرة أرقام digits مختلفة من 0 و حتى 9 و في الأعداد العشرية يمثل كل عمود قوة العدد 10 وبمعنى آخر كل عمود يمثل مرتبة أكبر من مرتبة العمود الذي على يمينه بعشر مرات و هكذا فإن القيمة العشرية 4623 مؤلفة من 3 آحاد و 2 عشرات و 6 مئات و 4 آلاف.

أما في علوم دنيا الحاسوب الواسعة و المتشعبة فتمثل الأعداد وفق التمثيل الثنائي binary و الذي يعتمد العدد 2 أساساً له و التمثيل الثماني octal و الذي يعتمد العدد 8 أساساً له و التمثيل الست عشري hexadecimal و الذي يعتمد العدد 16 أساساً له.

في النظام الست عشري كل عمود يُمثل قوة مختلفة للعدد 16 فمثلا العدد الست عشري التالي 4326 مؤلف من 6 آحاد و 2 ستة عشر و 3 مائتين و ست و خمسون و 4 أربعة آلاف و ست و تسعون :

$$(4326)_{16} = (\overset{3}{4}\,\overset{2}{3}\,\overset{1}{2}\,\overset{0}{6})_{16} = (6*16^0 + 2*16^1 + 3*16^2 + 4*16^3) = 6+32+768+16384 = (17190)_{10}$$

يتألف الأساس 16 من ستة عشر رقماً مختلفاً لتمثيل الأعداد وفق النظام الست عشري و هذه الأرقام هي من 0 .. 9 و الأحرف من A .. F (أو a .. f) لتمثيل الأرقام من 10 .. 15 على الترتيب و لذلك فإن القيمة الست عشرية CAFEتساوي بالنظام العشري:

$$(CAFE)_{16} = (\overset{3}{C}\,\overset{2}{A}\,\overset{1}{F}\,\overset{0}{E})_{16} = (12*16^3 + 10*16^2 + 15*16^1 + 14*16^0) = 49152+2560+240+14 = (51966)_{10}$$

و يمكننا استخدام القيم الست عشرية ضمن برامج التربو باسكال و للتمييز بين القيمة العشرية و القيم الست عشرية نضع رمز الدولار $ قبل القيمة الست عشرية فمثلاً القيمة الست عشرية $39 تساوي إلى القهمة العشرية 57 و البرنامج التالي يُوضح كيفية إلحاق القيم الست عشرية:

Code

```
program test;
var hexint:longint;
begin
hexint:=$CAFE;
writeln(' in decimal form ,hexint = ',hexint);
hexint:=$39;
writeln(' in decimal form ,hexint = ',hexint);
readln;
end.
```

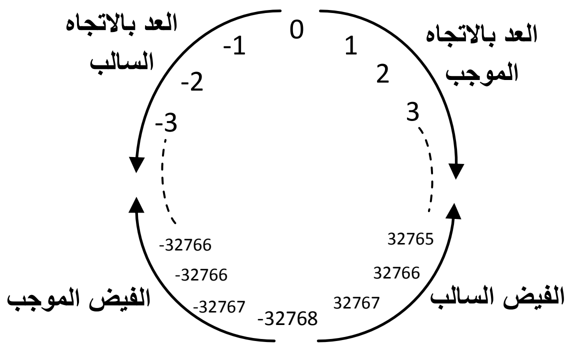
خرج البرنامج أعلاه:

```
in decimal form ,hexint = 51966
in decimal form ,hexint = 57
```

ملاحظة:

- الأعداد الصحية حالة خاصة من الأعداد الحقيقية و بشكل أدق هي أعداد حقيقية و لكن بدون أجزاء كسرية و لذلك يُمكن إلحاق قيمة صحيحة بمتحول حقيقي و بدون حدوث أخطاء في عملية الترجمة و لكن العكس غير صحيح أي لا يمكن إلحاق قيمة حقيقية بمتحول صحيح لأن المترجم سوف يعلن عن خطأ هو تضارب في أنواع المعطيات (و **بالتالي سوف يستخدم المترجم حق النقد الفيتو** ☺) .
- السؤال الذي قد يُطرح ماذا يحدث إذا حاولنا أن نمثل قيماً خارج المجال المسموح به لهذه القيم فمثلاً ماذا يحدث إذا جمعنا العدد 1 إلى العدد 32767.

حالة الفيض في الأعداد الكسرية كانت تسبب توقفاً في تنفيذ البرنامج أما الأعداد الصحيحة و بسبب طريقة تمثيلها فإن حالة الفيض لا تسبب خطأ في تنفيذ البرنامج و لكنها تُعطينا نتائج خاطئة (غير متوقعة) أي أننا إذا جمعنا العدد 1 إلى العدد 32767 فإن ناتج ضمن مترجم لغة باسكال سوف يعطينا -32768 و ليس 32768 و إن جمع العدد 2 إلى العدد 32767 ينتج لدينا -32767 و هكذا و ليس كما هو متوقع 32769 لأن هذا العدد يقع خارج المجال المسموح به للأعداد الصحية من نمط integer و بشكل مشابه إذا طرحنا العدد 2 من العدد -32768 نحصل على 32766 ومن فهم أجل معالجة حالة الفيض overflow ضمن الأعداد الصحيحة علينا أولاً أن نتخيل مجال الأعداد الصحيحة المسموحة مرتبة في حلقة كما في الشكل(المنطقي=التخيلي=الافتراضي) التالي:



المهندس خالد ياسين الشيخ

يحتل العدد 0 قمة الحلقة و القيم الموجبة تمتد على يمينه و القيم السالبة تمتد على يساره و في الأسفل بالإتجاه الموجب نجد القيمة الموجبة النهائية و هي 32767 و نجد بالاتجاه المعاكس القيمة السالبة النهائية -32768 و هاتان القيمتان متجاورتان ضمن الحلقة.

يمكن أن نتخيل أن عمليتي الجمع و الطرح تقومان بتوجيهنا بالاتجاه الصحيح ضمن هذه الحلقة إما يميناً في حالة عملية الجمع و إما يساراً في حالة عملية الطرح فلجمع عددين علينا إيجاد أولها ضمن الحلقة و بعدها نبدأ بعد الأعداد حسب قيمة العدد الثاني و باتجاه عقارب و القيمة اليي ستوقف عندها العد هي ناتج الجمع و لذلك فإن ناتج جمع 1 مع 32767 يُعطينا العد -32768 و بشكل مشابه إذا أردنا أن نطرح قيمة من أخرى فإننا نبدأ بالقيمة الأولى و نتحرك بعكس اتجاه عقارب الساعة و حسب القيمة الثانية حتى نصل إلى ناتج عملية الطرح هذه. فإذا طرحنا العدد 2 من العدد -32768 فسوف نجد ناتج هذه العملية هو 32766 .

و البرنامج التالي يُوضح لنا حالة الفيض ضمن الأعداد الصحيحة:

Code
program test;
const smallest=-32768;
largest=maxint;
var min,max:integer;
procedure getinteger(message:string; var value:integer);
begin

```
write(message,' ');
readln(value);
end;
begin
getinteger('subtract what from smallest? (0 to stop) ',min);
while(min<>0)do
begin
getinteger('add what to largest?',max);
write(smallest:7,' - ':5,min:7,' = ');
writeln(smallest-min:7);
write(largest:7,' + ':5,max:7,' = ');
writeln(largest+max:7);
getinteger('subtract what from smallest? (0 to stop) ',min);
end;
end.
```

خرج البرنامج السابق من أجل قيم للدخل 5 و -5 و 20:

```
subtract what from smallest? (0 to stop)  5
add what to largest? 5
 -32768   -       5 =   32763
  32767   +       5 =  -32764
subtract what from smallest? (0 to stop)  -5
add what to largest? -5
 -32768   -      -5 =  -32763
  32767   +      -5 =   32762
subtract what from smallest? (0 to stop)  20
add what to largest? 20
 -32768   -      20 =   32748
  32767   +      20 =  -32749
subtract what from smallest? (0 to stop)  0
```

المهندس خالد ياسين الشيخ

كما لاحظنا فإننا لم نحصل على أي رسالة خطأ عند حدوث الفيض لدى تنفيذ البرنامج و لذلك علينا توخي الحذر عندما نضرب أو نجمع قيما صحيحة كبيرة فإذا كنا نحتاج إلى التعامل مع اعداد كبيرة يجب علينا التعامل مع متحولات حقيقية أو مع متحول صحيح طويل.

نوع المعطيات عدد صحيح طويل type longint:

تزودنا لغة تربو باسكال بنوع المعطيات عدد صحيح longint و الذي يُمكننا من تمثيل مجال أكبر من القيم التي يمكن تمثيلها بواسطة نوع المعطيات عدد صحيح integer.

يحجز المترجم أربعة بايتات ما يعادل 32 bits لكل متحول من النوع الصحيح الطويل longint و هذا يُعطينا مجالاً للقيم يتراوح بين -2^{31} و $2^{31}-1$ أي بين -2147483648 و حتى 2147483647 و تزودنا لغة التربو باسكال بالثابت مسبق التعريف maxlongint و الذي يساوي أكبر قيمة يمكن تمثيلها من خلال نوع المعطيات longint .

و إذا حاولنا أن نُظهر أو نستخدم قيماً أكبر من الثابت maxlongint فإن الناتج سوف يعطينا حالة فيض overflow و هكذا فإن جمع العدد 1 إلى الثابت maxlongint سوف يعطينا العدد -2147483648 و معالجة حالة الفيض ضمن الأعداد الصحيحة الطويلة تتم بنفس الطريقة التي عولجت بها ضمن الأعداد الصحية integer و لكن هنا وفق مجال قيم أكبر. و البرنامج التالي يوضح لنا حدوث حالة الفيض من الأعداد الصحيحة الطويلة:

Code

```
program test;
const smallest=-maxlongint-1;
largest=maxlongint;
var min,max:longint;
```

```
procedure getlongint(message:string; var value:longint);
begin
write(message,' ');
readln(value);
end;
begin
getlongint('subtract what from smallest? (0 to stop) ',min);
while(min<>0)do
begin
getlongint('add what to largest?',max);
write(smallest:7,' - ':5,min:7,' = ');
writeln(smallest-min:7);
write(largest:7,' + ':5,max:7,' = ');
writeln(largest+max:7);
getlongint('subtract what from smallest? (0 to stop) ',min);
end;
end.
```

خرج البرنامج السابق من أجل قيم للدخل 1 و 2- و 5 و 20:

```
subtract what from smallest? (0 to stop)  1
add what to largest? 1
 -2147483648    -            1 =   2147483647
  2147483647    +            1 =  -2147483648
subtract what from smallest? (0 to stop)  -1
add what to largest? -2
 -2147483648    -           -1 =  -2147483647
  2147483647    +           -2 =   2147483645
subtract what from smallest? (0 to stop)  5
add what to largest? 5
 -2147483648    -            5 =   2147483643
  2147483647    +            5 =  -2147483644
subtract what from smallest? (0 to stop)  20
add what to largest? 20
 -2147483648    -           20 =   2147483628
  2147483647    +           20 =  -2147483629
subtract what from smallest? (0 to stop)  0
```

المهندس خالد ياسين الشيخ

نوع المعطيات كلمة type word:

يُخصص نوع المعطيات كلمة word المعرف في لغة التربو باسكال لكل عدد كلمة ثنائية في الذاكرة مكونة من بايتين (16bits) و هو يُستخدم لتمثيل القيم الموجبة فقط و مجال الأعداد في نوع المعطيات كلمة يتراوح بين 0 و 65535 أي بين 0 و $2^{16}-1$ و الفرق بين نوع المعطيات عدد صحيح و نوع المعطيات كلمة word هو أن الخانة اليسرى leftmost bit تستخدم في الحسابات إذا كان العدد من النوع word و تستخدم لتمثيل إشارة العدد إذا كان العدد من النوع integer و حلة الفيض ضمن القيم من النوع word تُعالج بشكل مشابه لما هي عليه في integer مع عدم وجود قيم سالبة ضمن الحلق و أن القيمتان 0 و 65535 متجاورتان ضمن الحلقة.

و البرنامج التالي يُرينا حالة فيض في متحول من النوع word:

Code
program test; var w:word; begin

```
w:=65535;
w:=w+1;
writeln(w);
w:=w+1;
writeln(w);
readln;
end.
```

خرج البرنامج السابق هو:

```
0
1
```

نوع المعطيات عدد صحيح قصير shortint type:

هذا النوع يُخصص له حجرة ذاكرة واحدة أو بايتاً واحداً لكل عدد حيث يُمثل نوع المعطيات عدد صحيح قصير الأعداد من 128-وحتى 127 و في هذا النوع يمثل البت الأعلى أو الخانة اليسرى إشارة العدد و حالة الفيض تعالج بشكل مشابه لما هو الحال في integer و لكن مع مجال قيم أصغر.

و البرنامج التالي يوضح لنا حالة حدوث فيض في متحول صحيح قصير:

Code

```
program test;
var s:shortint;
begin
s:=125;
s:=s+128;
writeln(s);
s:=-128;
s:=s-270;
writeln(s);
readln;
end.
```

خرج البرنامج السابق هو:

```
-3
114
```

ملاحظة: ناتج جمع s=s+128 هو 253 و نلاحظ أن هذا الناتج قد تجاوز قيم التي يمكن منحها للمتحول s و الدوران بجهة عقارب الساعة فيكون لدينا: 3-=253-256 .

و ناتج جمع s=s-270 هو 398- و هذا الناتج قد تجاوز القيم التي يمكم منحها للمتحول s و الدوران هنا بعكس عقارب الساعة فيكون : 142-=398+256- وأيضاً القيمة 142- خارج المجال المسموح به ويتم أيضاً 114=142+256-

نوع المعطيات بايت byte type:

يخصص له حجرة ذاكرة واحدة أو بايت لكل متحول و لا يستخدم بتاً للإشارة أي يمكنه تمثيل الأعداد الموجبة من0 و حتى 255 أي من 0 وحتى 2^8-1 و نوع المعطيات مفيد جداً عند قراءة ملف مكتوب بالشفرة الثنائية.

نوع المعطيات رمز(حرف) the char type:

يستخدم نوع المعطيات رمز char في لغة باسكال رموز مستقلة على شكل شفرات ASCII و المتحولات من نوع char يُخصص لها بايت واحداً من الذاكرة و هذا يعني أن شفرات الرموز تتراوح بين بين 0 و 255 و من أجل تضمين البرنامج رمزاً معيناً يجب أن نضع هذا الرمز ضمن فاصلتين علويتين فمثلا العبارة التالية تُلحق الحرف Q بالمتحول C الذي ينتمي للنوع char :
C:='Q';
تضمن مجموعة رموز ASCII القياسية 128 رمزاً و مجال شفراتها يقع بين 0 و 127 أما الشفرات من 128 و حتى 255 فتُعرف على أنها شفرات رموز ASCII الموسعة extended ASCII characters و يمكن استخدام هذه الشفرات لأغراض خاصة ضمن البرامج.
كل رمز هجائي ضمن مجموعة رموز ASCII يتطابق مع شفرة خاصة به إذ يملك الحرف A شفرة ASCII المقابلة له و هي 65 أما الحرف a فيملك شفرة ASCII مقابلة له و هي 97 و شفرات ASCII للأحرف مرتبة بشكل تسلسلي أي سفرة ASCIIللحرف A هي 65 و للحف B هي 66 و هكذا حتى نصل إلى الشفرة 90 و هي الشفرة المقابلة للحرف z . و بشكل مشابه فإن الأحرف الصغيرة تملك شفرات ASCII مرتبة أيضاً ابتداء من شفرة الحرف a و هي 97 و حتى شفرة الحرف z و هي 122.
و شفرات ASCII المقابلة للأرقام مرتبة ابتداء من 48 و هي الشفرة المقابلة للرقم 0 و حتى 57 و هي الشفرة المقابلة للرقم 9 .
و لكن يجب الانتباه هنا إلى أن الأرقام كرموز تختلف عن الأرقام كأعداد فمثلا: الصيغتان التاليتين مختلفتان تماماً 3 و '3' حيث '3' هي قيمة رمز أما 3 فهي قيمة صحيحة .
بالإضافة إلى بعض الرموز الخاصة مثل : (# و & و رمز الفراغ ....)
ومز ASCII هذه تدعى بالرموز القابلة للطباعة printable characters و هذه الرموز تملك شفرات ASCII مقابلة من 32 و حتى 126 و يمكن أن نعين هذه الرموز باستخدام شفراتها المقابلة مباشرة إذا سبقها الرمز # فمثلا الصيغة #67 تمثل الحرف C و الصيغة #106 تمثل الحرف j .
و توجد مجموعة من الرموز و التي تدعى رموز التحكم control characters و كل رمز من هذه الرموز يقوم بعمل معين أو بإرسال رسالة محددة فعلى سبيل المثال : رمز مفتاح الجدولة TAB key هو رمز تحكم و رمز إرجاع الحاملة CR(carriage return) هو رمز تحكم و رمز التغذية السطرية LF(line feed) حيث كل الرموز التي شفراتها من 0 و حتى 31 هي رموز تحكم ويمكننا تعيين رمز التحكم عن طريق استخدام الرمز # متبوعاً بشفرة الرمز المُراد استخدامه فمثلاً الصيغة #7 تعني شفرة ASCII ذات الرقم 7 و التي تعطي صوت الجرس .
و شفرات ASCII من 1 و حتى 26 تملك كلها أسماء تتضمن أحرف فمثلا الشفرة رقم 1 تعرف على أنها CTRL-A و هكذا حتى الشفرة ذات الرقم 26 و التي تعرف على أنها ctrl-z و حتى نعين أي من هذه الرموز يجبأن نسبقها بالرمز ^ فمثلا الصيغة ^G تمثل الرمز الذي شفرته المقابلة و هي 7 و هو صوت الجرس و الصيغة ^I تمثل الرمز الذي شفرته المقابلة هي 9 و هو مفتاح الجدولة.
و البرنامج التالي يبين كيفية تخيص أنواع

Code

```
program test;
const beepchar=#7;
    beepchar2=^G;
    Tabchar=^I;
var val:char;
procedure getchar(message:string; var value:char);
begin
write(message,' ');
readln(value);
end;
begin
getchar('enter a charactr please.',val);
writeln(beepchar,tabchar,val,tabchar,val,beepchar2);
readln;
end.
```

خرج البرنامج السابق من أجل إدخال أي محرف هو (مع إصدارصوت):

```
enter a charactr please. A
        A       A
```

و يمكن تطبيق معامل الإلحاق و معاملات المقارنة في لغة باسكال على المتحولات من النوع char و في أي عملية مقارنة تستخدم شفرات ASCII المقابلة للمعطيات من النوع char.
فمثلا:

'H'<'h'

ناتج المقارنة السابق هو true لأن الشفرة المقابلة للرمز H هي 72 و هي أصغر من الشفرة المقابلة للرمز h و هي 104.

و نستطيع جمع الرموز باستخدام معامل الجمع + و ينتج لدينا سلسلة رمزية string فمثلا ناتج التعبير التالي هو:

Str='S'+'A';

هو 'SA'.

نوع المعطيات سلسلة رمزية the string type:

متحولات هذا النوع تُستخدم لتخزين تتابع من الرموز و يمكن التصريح او التعريف عن سلاسل رمزية بدون تحديد لطول هذه السلاسل أو بتحديد لطولها الأعظمي.
مثال:

```
Var     defaultstr:string;
         Str30: string[30];
         Str10: string;
```

حيث التصريح السابق لثلاثة متحولات من النوع سلسلة رمزية حيث صُرِّح عن المتحول الأول defaultstr كمتحول من النوع string و بدون تحديد لطول هذا المتحول و هذا أنه بلغة التربو باسكال لا يمكن أن يملك أمثر من 255 رمزاً و اما التصريح الثاني فقد حدد أن المتحول str30 من النوع string و لكن بطول أعظمي قدره 30 رمزا (بايت) و كذلك المتحول الثالث str10 ولكن بطول قدره 10 رموز.
و حتى نعطي قيما للمتحولات من النوع string علينا وضع سلسلة من الرموز وفق تتابع معين و ضمن فاصلتين علويتين فمثلا:

Str30:='syria arabic' ;

إذا حاولنا أن نُلحق سلسلة رمزية إلى متحول و كان طول هذه السلسلة أكبر من حيز التخزين المخصص لهذا المتحول فإن الرموز الزائدة يتم تجاهلها و هكذا فإن المترجم لن يُعارض.
و يمكن تطبيق معامل الإلحاق و معاملات المقارنة على نوع المعطيات string و تستخدم شفرات ASCII في عمليات المقارنة تلك و تُنجز هذه المقارنة رمزا رمزا فمثلا المقارنة التالية تعطي القيمة true:

'helpful'>'hello'

لأن الحرف p ضمن السلسلة الرمزية helpful يملك شفرة ASCII أكبر من شفرة الحرف l ضمن السلسلة الرمزية hello واما ما قبل هذه النقطة فالسلسلتين متطابقتان تماماً.
ويمكن استخدام معامل الجمع لأنتاج سلسلة أطول مثال:

Defaultstr:='syria'+' arabic';

تصبح قيمة المتحول defaultstr هو 'syria arabic' .

التحكم بشكل الخرج controlling output format:

بشكل مشابه من خلال دراستنا للأعداد الحقيقية و الأعداد الصحيحة نستطيع تحديد شكل خرج خاص للرموز و السلاسل فمن أجل تحديد عرض الحقل الخاص بالمعلومات الرمزية الواجب كتابتها نضع النقطتان : و نكتب بعدها عرض الحقل الذي نريد فإذا كانت قيمة الخرج أكبر من عرض الحقل المخصص لها يتم تجاهل لعرض الحقل وستكتب المعلومات كاملة إما إذا كانت السلسلة أقصر من عرض الحقل المخصص لها فستكتب المعلومات اعتباراً من آخر عمود في عرض الحقل(رصف يميني).

يمكن أن تحتوي السلسلة الرمزية على أي رمز من رموز Ascii و بالتالي عند طلب إظهار السلسلة باستخدام الإجراء write أو writeln سوف تظهر الرموز كما هي.و لكن هنالك نقطة توقع المترجم في إرباك و هي عندما تحتوي السلسلة على فاصلة علوية حيث أن الإجراءين write و writeln يستخدمان الفاصلة العلوية ' لتحديد بداية و نهاية السلسلة الرمزية
مثال:

Writeln('that 's your disk');

حيث العبارة السابقة سوف تسبب خطأ في الترجمة و لن يقبلها المترجم لأنه يتوقع وجود قوس الإغلاق) بعد الفاصلة العلوية في الكلمة ' this و الحل استبدال الفاصلة العلوية بفاصلتين علويتين متتاليتين بدون فراغ بينهما لإخبار المترجم بأن هذه الفاصلة ليست لإنهاء السلسلة الرمزية و إنما هي رمز من رموز السلسلة الرمزية كما يلي:

Writeln('that ''s your disk');

وعندها تظهر العبارة this ' s you disk على الشاشة.

أنواع المعطيات المنطقية logical types:

تملك لغة باسكال نوعاً مسبق التعريف لتخزين القيم المنطقية و التي يمكن أن تكون إما false او true.حيث تملك لغة باسكال ثابتين مسبقي التعريف هما true و false و هما يعودان إلى نوع المعطيات البولياني Boolean و البرنامج التالي يبين لنا التصريح عن المتحولات البوليانية و كيفية استخدامها.

Code
```
program test;
const maxtrials=10;
var bool1,bool2:boolean;
count,val1,val2:integer;
begin
randomize;
for count:=1 to maxtrials do
begin
val1:=random(maxint);
val2:=random(maxint);
if(val1>val2)then
bool1:=true
else
bool1:=false;
if(val1-val2) mod 2=0 then
bool2:=true
else
bool2:=false;
writeln(val1:8,' ',val2:8,' ',bool1:8,' ',bool2:8);
end;
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
    6084    28177    FALSE    FALSE
    6690     5560     TRUE     TRUE
   11932     4921     TRUE    FALSE
    9576     1033     TRUE    FALSE
   31717    18820     TRUE    FALSE
    3354    27640    FALSE     TRUE
   13494    19615    FALSE    FALSE
   22179     4114     TRUE    FALSE
   19228    29058    FALSE     TRUE
    4216     5261    FALSE    FALSE
```

المهندس خالد ياسين الشيخ

نستطيع إظهار قيم بوليانية و للكن لا نستطيع إدخال قيم بوليانية بواسطة الإجراء read أو readln (لا نستطيع قراءة متحول أو قيمة بولياني من لوحة الدخل القياسي (لوحة المفاتيح) أو ملف و الطريقة الوحيدة لإلحاق قيمة بوليانية هي أن نضع تعبيراً بولياناً boolean expression في الطرف الأيمن من معامل الإلحاق حيث يعطي التعبير البولياني إحدى القيمتين المنطقيتين:إما true أو false و التعبير المتضمن معامل مقارنة يستخدم دائماً كتعبير بولياني فعلى سبيل المثال تلحق التعابير التالية قيماً بوليانية بمتحولات بوليانية:

```
Bool1:=7.82>13            (*assign false to bool1*)
Bool1:='a'='a'            (*assign true to bool1 *)
Bool1:=maxint<maxlongint; (*assign true to bool1 *)
```

المعاملات المطبقة على المتحولات البوليانية operators for Boolean variables:

يمكننا استخدام معامل الإلحاق و معاملات المقارنة مع القيم البوليانية فعند مقارنة قيمتين بوليانيتين يجب الانتباه إلى أن FALSE<TRUE و هكذا فإن العبارة التالية تعطينا القيمة TRUE:

TRUE>FALSE

لكن مثل هذه المقارنات **المباشرة** بين القيم البوليانية تكون عادة غير ضرورية و نادرة الاستخدام بالإضافة إلى هذه المعاملات يوجد بعض المعاملات التي تطبق مباشرة على الحدود البوليانية و هي and و or و xor و not و تدعى هذه المعاملات بالمعاملات المنطقية.

المعامل AND the AND operator:

يأخذ المعامل حدين بوليانيين و يعيد قيمة بوليانية و هذا المعامل يعيد القيمة true فقط إذا كان قيمة كلا الحدين اليمني و اليساري TRUE.

و سلوك هذا المعامل يُمكن أن يُلخص بجدول الحقيقة true table الذي يبين لنا ناتج كل حالة من الحالات الممكنة لزوج الحدود البوليانية:

ناتج تطبيق المعامل:A AND B	قيمة الحد اليميني B	قيمة الحد اليساري A
False	False	False
False	True	False
False	False	True
True	True	True

و يجب الانتباه هنا إلى أن الأقواس حول التعابير البوليانية ضرورية جداً بسبب كون المعامل and يملك أسبقية أعلى من أسبقية معاملات المقارنة (و قد يؤدي عدم وضع الأقواس إلى حدوث أخطاء منطقية).

المعامل or The OR operator:

يأخذ المعامل OR حدين بوليانيين و يعيد قيمة بوليانية و تكون قيمة هذا الناتج FALSE فقط عندما يكون كلا الحدين false و إلا فإنه يأخذ دائماً القيمة True (أي أن المعامل or يعيد القيمة true إذا كان أحد أو كلا الحدين لهما القيمة true) و جدول الحقيقة الواصف للمعامل OR هو التالي:

ناتج تطبيق المعامل:A AND B	قيمة الحد اليميني B	قيمة الحد اليساري A
False	False	False
True	True	False
True	False	True
True	True	True

يملك المعامل or أسبقية أعلى من أسبقية معامل الإلحاق و معاملات المقارنة ولذلك يجب الانتباه إلى وضع أقواس حول التعابير البوليانية التي تحتوي على المعامل or و فكرة استخدام الأقواس دائماً فكرة ممتازة (على الأقل من من أجل أن نتأكد نحن و المترجم من أننا نسير باتجاه واحد).

المعامل xor the XOR Operator:

المعامل xor يعطينا القيمة TRUE فقط إذا كانت قيمة أحد حديه True . وجدول الحقية لهذا المعامل هو:

ناتج تطبيق المعامل:A AND B	قيمة الحد اليميني B	قيمة الحد اليساري A
False	False	False
True	True	False
True	False	True
False	True	True

يختلف هذا المعامل عن المعامل or فقط في الحالة الأخيرة من جدول الحقيقية (حيث أن المعامل or يُعطي القيمة true إذا كانت قيمة كلا الحدين true ي حين أن المعامل xor يُعطي القيمة false).

ملاحظة هامة : عكس xor هو xor (فائدة هذه الملاحظة في التشفير (مادة أمن المعلومات)).

المعامل Not the NOT Operator:

المعامل البولياني NOT يأخذ حداً بولياناً واحداً و يعيد قيمة بوليانية واحدة معاكسة لقيمة حده الوحيد و هكذا إذا كانت قيمة الحد true فإن المعامل Not يُعطي القيمة false.
يمكننا تطبيق المعامل not على أي قيمة بوليانية أو أي متحول متحول بولياني أو أي تعبير بولياني.
فمثلاً:
لدينا عبارات التالية صحيحة:

```
Not not false             (* تُعطي false *)
Not true                  (* تُعطي false *)
Not (10>20)               (* تُعطي true *)
Not ((3>5)and (5>7))      (* تُعطي true *)
Not(3>5) and (5>7)        (* تُعطي false *)
Not(3>5) or (5>7)         (* تُعطي true *)
```

لدينا عبارات خاطئة:

```
Not (3)            (* لا يمكن استخدام not مع الأعداد الصحيحة *)
Not('g')           ( لا يمكن استخدام not مع الرموز *)
Not('good bye')    (لا يمكن استخدام not مع السلاسل الرمزية)
```

يجب الانتباه إلى أن المعامل not يملك أسبقية على المعاملات الأخرى (and و or و xor) و لذلك فإن التعبير الخامس و السادس في العبارات الصحيحة (3>5) not يحسب أولا و قيمته true و لذلك يجب الانتباه إلى عملية وضع الأقواس لأجل معرفة أولوية العمليات.

أسبقية المعاملات البوليانية precedence for the Boolean operator :

كل المعاملات البوليانية تملك أسبقية أعلى من أسبقية معاملات المقارنة و معامل الإلحاق و الجدول التالي يُظهر لنا ترتيب المعاملات من حيث أسبقيتها:

نوع المعاملات	المعاملات
النفي المنطقي	not
معاملات الضرب و القسمة	*,/,div,mod,and
معاملات الجمع و الطرح	+,-,or,xor
معاملات المقارنة	=,<>,>,>=,<,<=
معامل الإلحاق	:=

أنواع المعطيات التعدادية enumerated types:

تمكننا لغة باسكال من إنشاء أنواع معطيات بسيطة بواسطة سردها أو تعدادها و بسبب طبيعة نوع المعطيات يُعرف هذا النوع بالنوع التعدادي enumerated type حيث يتم سرد أو تعداد جميع القيم التي تنتمي لهذا النوع من المعطيات فعلى سبيل المثال: افترض=تخيل أننا نريد تمثيل أوراق اللعب عندئذ نحتاج إلى تمثيل مجموعة أرقام هذه الأوراق و مجموعة لتمثيل أنواع هذه الأوراق و يبين البرنامج التالي كيفية إنشاء متحولات لتمثيل هذه المعلومات:

Code

```
program test;
const maxtrials=10;
type
cardsuit=(club,diamond, heart, spade);
facevalue=(tow, three, four , five, six, seven, eight, nine
        ,ten,jack,gueen,king,ace);
var suit:cardsuit;
   face:facevalue;
begin
suit:=club;
writeln(ord(suit));
face:=ten;
writeln(ord(face));
```

```
suit:=cardsuit(0);
writeln(ord(suit));
writeln(succ(ord(jack)));
readln;
end.
```

خرج البرنامج أعلاه هو:

```
0
8
0
10
```

استخدام التعريف type لإنشاء أنواع المعطيات التعدادية

:using a type definition to create enumerated type

أول خطوة من خطوات العمل مع الأنواع التعدادية هي تعريفها ضمن قسم type و ذلك في قسم التصريح الخاص بالبرنامج ومن خلال هذا التعريف يحدد اسم لهذا النوع و تسرد جميع القيم التي تنتمي إليه ضمن لائحة توضع بين قوسين صغيرين و يتم الفصل بين قيم هذا النوع بفاصلة (,) و الصيغة الكتابية لإنشاء نوع تعدادي ضمن قسم التعريف type هي:

اسم النوع = (قيمة 1, قيمة 2 , ...,قيمة n)

إشارة المساواة و الأقواس ()= هي رموز **ثابتة** عند تعريف كل نوع أما العناصر المتبقية في هذه الصيغة فيمكن تغييرها مع كل تعريف و يجب الانتباه إلى أن قيم هذا النوع لا توضع ضمن أقواس علوية ' ' لأن هذه القيم ليست سلاسل رمزية و لكنها أسماء يُخصص المترجم لها شفرات داخلية حيث يخصص مترجم لغة تربو باسكال العدد 0 حيث أن القيم تأخذ شفراتها حسب العلاقة الرياضية التالية :

<موقع هذه القيمة ضمن اللائحة > 1 -

مما يسهل على المترجم التعامل مع هذه القيم و الترتيب الُمستخدم في سرد هذه القيم يُعتبر ترتيباً فعلياً لهذه القيم فالعبارة التالية تعطينا القيمة true اعتماداً على تعريفات البرنامج السابق:

Heart>club;

و السبب في ذلك أن الشفرة الداخلية للقيمة heart هي 2 في البرنامج السابق و أما الشفرة الداخلية للقيمة club هي 0 و بالتالي فإن 0<2 و بشكل مشابه فإن القيمة ace أكبر من القيمة jack و هذا الترتيب مفيد جداً في البرنامج فمن اجل مقارنة ورقتين من أوراق اللعب نستخدم معاملات المقارنة لنقرر أي القيمتين أكبر.

لا يمكننا تطبيق أي معامل على أنواع المعطيات هذ سوى معامل الإلحاق و معاملات المقارنة و يُمكننا تعريف أكثر من نوع تعدادي حيث يعرف النوع الأول مباشرة بعد كلمة type و الأسطر الأخرى يمكن أن تُعرف على الأسطر التالية لها أو على نفس السطر .

التعريف type هو واصف للمعطيات و هذا التعريف لا يُنشئ أي متحولات لأنواعه التي يُعرفها و لا يُخصص أي مساحات لتخزين المتحولات التي تنتمي لأنواعه،

حيث لم يكن هنالك أي متحول من النوع cardsuit حتى صرحنا عن المتحول suit على أنه من هذا النوع و بالمقابل لم يُخصص أي مكان لتخزين قيمة أي متحول من النوع facevalue حتى صرحنا عن المتحول face على أنه ينتمي لهذا النوع التعدادي فعندما نعرف نوعاً تعدادياً يمكننا التصريح عن متحولات تنتمي لهذا النوع و للقيام بذلك نستخدم نفس الصيغة الكتابية المستخدمة في التصريح عن المتحولات التي تنتمي لأنواع معطيات أخرى و كمثال على ذلك صرحنا عن المتحولين suit و face بعد أن عرفنا أنواعهما في البرنامج السابق.

وحجم الذاكرة المخصص لكل متحول من النوع التعدادي يعتمد على عدد القيم التي يملكها هذا النوع ففي الأنواع التي تملك قيماً مختلفة أقل من 256 قيمة يخصص لها بايت واحد أما الأنواع التي تملك قيماً أكبر فُيخصص لها بايتين.

وعندما نصرح عن المتحولات من النوع التعدادي يمكننا تطبيق معامل الإلحاق أو معاملات المقارنة على هذه المتحولات ولسوء الحظ لا يمكننا كتابة هذه القيم مباشرة على الشاشة أو على ملف و لا يمكننا قراءتها من لوحة المفاتيح أو من ملف مباشرة.

المعطيات المركبة هي التي تستخدم لتخزين حدود متعددة من المعلومات ضمن متحول واحد وأنواع المعطيات هذه هي array و record و set و التي تعرف بأنواع المعطيات المركبة structured types أو أنواع المعطيات المجمعة aggregate types لأن متحولات هذه الأنواع تتألف من عدة عناصر.
و لدينا أيضاً نوع المعطيات كائن object و الذي يستخدم في البرمجة كائنية(غرضية) التوجه object –oriented programming
يتألف نوع المعطيات نسق array من مجموعة من العناصر تدعى بالحجرات cells و هذه الحجرات تحتوي على معلومات تنتمي إلى نوع معطيات واحد أما نوع المعطيات سجل record فيتألف من عنصر أو أكثر يُدعى كل واحد منها حقلاً field ينتمي كل منهم إلى نفس نوع المعطيات أو إلى أنواع معطيات مختلفة و المجموعة set تتألف من مجموعة غير مرتبة من القيم تدعى بالعناصر elements و تنتمي لنفس نوع المعطيات.
تستخدم أنواع المعطيات المركبة هذه في بناء كتل لإنشاء أنواع معطيات مناسبة لكل مسألة برمجية فعلى سبيل المثال:

- إذا أردنا تخزين معدلات مجموعة من الطلبة نستخدم من أجل ذلك نسقاً من الأعداد الحقيقية.
- إذا اردنا تخزين نبذة عن طالب واحد مثلاً:اسمه و معدله و اختصاصه و عمره.... إلخ نستخدم لأجل ذلك سجلاً لتخزين هذه البيانات.
- إذا أردنا تخزين نبذة عن مجموعة من الطلاب فستخدم نسقاً كل حجرة من حجراته عبارة عن سجل.
- إما إذا أردنا أن نخزن المواد التي نجح فيها الطالب فإننا نستخدم مجموعة set لتمثل هذه اللائحة من المواد.
- إذا أردنا أن نُضمن النبذة عن الطالب بالمواد الناجح فيها فإننا نستخدم سجلاً بحيث يحتوي هذا السجل على السجل السابق الذي يحتوي على موجز عن حياة الطالب و مجموعة فيها لائحة بالمواد الي نجح فيها هذا الطالب.

تعريفات و تصريحات definitions and declarations:

عندما نصرح عن متحول على أنه ينتمي إلى أحد أنواع المعطيات البسيطة فإن المترجم compiler يعرف تماما ماذا يعني هذا التصريح.أي أن المترجم يعرف عدد حجرات الذاكرة التي يجب أن يُخصصها لهذا المتحول و يعرف أيضاً العمليات المسموح تطبيقها على هذا المتحول و لكن عندما نعمل مع أنواع معطيات مركبة فإن الأمر سوف يختلف قليلاً لأن أنواع المعطيات المركبة تبنى من كتل blocks و هذه الكتل المتنوعة يمكن أن تكون بحد ذاتها أنواع معطيات أخرى و لهذه الكتل أنواع هي:

1) الكتل المتجانسة و تمثلها الأنساق (المصفوفات) arrays.
2) الكتل المتغايرة(غير المتجانسة) و تمثلها السجلات records.
3) الكتل ذات المحتويات القابلة للزيادة أو النقصان حسب الحاجة و تمثلها المجموعات sets.

فعندما ننشئ نوع معطيات مركب فإننا فعلياً نركب عدة كتل أساسية ضمن التصريح عن نوع المعطيات هذا إذ يمكن بناء نوع معطيات مركب عبارة عن نسق كل حجرة من حجره عن سجل ما و نوع المعطيات هذا يجب أن يصرح عنه في قسم TYPE و الشكل العام للتصريح عن أنواع المعطيات المركبة هو:

```
✓ مميز=array[حدود أدلة النسق] of نوع مطيات أساسي ;
✓ مميز=record
   لائحة من الحقول
   End;
✓ مميز=set of نوع معطيات أساسي ;
✓ مميز=object
   لائحة من الحقول
  End;
```

نوع المعطيات نسق Array the array type:

لنفترض = لنتخيل أننا نريد حساب تكرار الأعداد العشوائية الصحيحة بين 0 و 99 و التي يولدها مولد الأعداد العشوائية في لغة باسكال فلتمثيل 100 حالة يجب أن نصرح عن 100 متحول مختلف لتمثل القيم العشوائية المحتمل توليدها و لتكن هذه f99...f2-f1-f0 و بعدها تستخدم مثلاً عبارة case مع 100 حالة لفحص العدد المتولد وزيادة قيمة المتحول لخاص به. و عبارة case هذه ستكون كالتالي:

```
Case Remainder of :
0: inc(f0);
1: inc(f1);
2: inc(f2);
(*
        And so on, to *)
99: inc(f99);
End;
```

ولكن الخطوات السابقة صعبة و تطيل البرنامج في حين أننا نستطيع استخدام المصفوفات للتعامل مع مثل هذه لحالات حيث يعرف النسق بأنه عبارة عن متحول يحتوي على على عدة قيم متجمعة فيما بينها و تنتمي إلى نوع معطيات واحد و هذه القيم المستقلة تدعى بحجرات cell النسق و تملك كل حجرة دليل index مرتبطاً بها.

التعريف و التصريح عن النسق defining and declaring an array:

يمكن حل مشكلة فرز الأعداد الصحيحة العشوائية المتولدة بتمثيلها وفق نسق يحتوي على 100 عنصر كل حجرة من حجره تحتوي على عدد صحيح و البرنامج التالي يوضح ذلك:

Code

```
program testDigits;
const maxtrials=5000;
type
remainders=array[0..99] of integer;
var freqs:remainders;
index,result:integer;
procedure initremainders(var vals:remainders; initvalue:integer);
var index:integer;
begin
for  index:=0 to 99 do
vals[index]:=initvalue;
end;
procedure dispremainders(vals:remainders);
var index:integer;
begin
for index:=0 to 24 do
begin
write('[':5,index:3,']: ',freqs[index]:3);
write('[':5,index+25:3,']: ',freqs[index+25]:3);
write('[':5,index+50:3,']: ',freqs[index+50]:3);
write('[':5,index+75:3,']: ',freqs[index+75]:3);
writeln;
end;
readln;
end;
begin
randomize;
initremainders(freqs,0);
for index:=1 to maxtrials do
begin
result:=random(maxint)mod 100;
freqs[result]:=freqs[result]+1;
end;
dispremainders(freqs);
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
[  0]:  56    [ 25]:  54    [ 50]:  51    [ 75]:  41
[  1]:  41    [ 26]:  50    [ 51]:  50    [ 76]:  52
[  2]:  46    [ 27]:  51    [ 52]:  57    [ 77]:  35
[  3]:  43    [ 28]:  44    [ 53]:  50    [ 78]:  55
[  4]:  42    [ 29]:  41    [ 54]:  53    [ 79]:  58
[  5]:  48    [ 30]:  43    [ 55]:  46    [ 80]:  51
[  6]:  43    [ 31]:  52    [ 56]:  52    [ 81]:  56
[  7]:  49    [ 32]:  41    [ 57]:  59    [ 82]:  44
[  8]:  46    [ 33]:  59    [ 58]:  56    [ 83]:  52
[  9]:  51    [ 34]:  49    [ 59]:  58    [ 84]:  54
[ 10]:  44    [ 35]:  47    [ 60]:  53    [ 85]:  52
[ 11]:  49    [ 36]:  43    [ 61]:  38    [ 86]:  49
[ 12]:  56    [ 37]:  51    [ 62]:  54    [ 87]:  53
[ 13]:  55    [ 38]:  48    [ 63]:  55    [ 88]:  42
[ 14]:  51    [ 39]:  54    [ 64]:  54    [ 89]:  56
[ 15]:  48    [ 40]:  50    [ 65]:  50    [ 90]:  46
[ 16]:  47    [ 41]:  55    [ 66]:  57    [ 91]:  48
[ 17]:  47    [ 42]:  51    [ 67]:  54    [ 92]:  49
[ 18]:  44    [ 43]:  49    [ 68]:  56    [ 93]:  48
[ 19]:  50    [ 44]:  53    [ 69]:  40    [ 94]:  36
[ 20]:  60    [ 45]:  54    [ 70]:  47    [ 95]:  60
[ 21]:  48    [ 46]:  51    [ 71]:  41    [ 96]:  56
[ 22]:  50    [ 47]:  53    [ 72]:  67    [ 97]:  50
[ 23]:  46    [ 48]:  46    [ 73]:  54    [ 98]:  49
[ 24]:  56    [ 49]:  45    [ 74]:  62    [ 99]:  44
```

نلاحظ من خلال البرنامج السابق أنه تم تعريف النسق ضمن قسم التعريف type و هذا حال جميع أنواع المعطيات المركبة :

Type remainders=array[0..99] of integer;

هذه العبارة تُعرف نوع معطيات جديدة تُطلق عليه الاسم remainders و نوع المعطيات هذا مبني من عناصر معروفة و في حالتنا هذه مبني من نسق و النسق يتألف من عناصر تنتمي لنفس نوع المعطيات و لذلك يجب عند تعريف النسق أن نحدد نوع المعطيات و الذي ندعوه نوع المعطيات الأساسي base type و في المثال السابق كان نوع المعطيات الأساسي للنسق remainders هو نوع المعطيات عدد صحيح integer و هذا يعني أن كل حجرة من حجرات النسق يخصص لها بايتين من الذاكرة حيث كل حجرة عبارة عن متحول من النوع الصحيح و يمكن أن تستخدم كما يُستخدم المتحول و تعامل كما يُعامل المتحول الصحيح.

وبعد أن حددنا نوع المعطيات الأساسي للنسق علينا تحديد طول النسق من خلال وضع مجال جزئي ضمن قوسين لتحديد ادلة حجرات النسق و في مثال السابق كانت حدود النسق bounds هي [0..99] أي أن الحجرة الأولى في النسق تسمى الحجرة رقم 0 و آخر حجرة من حجرات النسق تدعى بالحجرة رقم99. ومن خلال حدود النسق يجري إخبار المترجم بأن النسق المذكور يحتوي على 100 حجرة من النوع الصحيح.

الوصول إلى حجرات النسق المستقلة accessing individual array cells:

نستطيع الوصول إلى حجرات النسق عن طريق اسم النسق freqs متبوعاً (و بين قوسين مربعين) بدليل الحجرة و دليل الحجرة لا يمكن أن يكون عدد حقيقي فمثلاً العبارة التالية :

freqs[result]:=freqs[result]+1;

تزيد العبارة السابقة محتوى الحجرة الي دليلها يطابق قيمة المتحول result من النسق freqs بمقدار 1 و يمكن الاستعاضة عن العبارة السابقة بالعبارة التالية التي تقوم بنفس العمل تماماً :

INC(freqs[result]);

حيث يشير التركيب freqs[result] إلى موضع في الذاكرة يحتوي على قيمة من النوع الصحيح integer ولذلك قبلها الإجراء INC كمتحول وسيطي argument له.

دليل حجرة النسق مقابل قيمة هذه الحجرة cell index versus cell value:

ينبغي دائما التمييز بين دليل الحجرة وبين القيمة المخزنة في هذه الحجرة إذ إن دليل الحجرة هو جزء من اسم الحجرة أي التركيب freqs[5] يمثل موضعاً محدداً في الذاكرة يحتوي على عدد صحيح و هذا العدد يمثل عدد مرات تكرار الرقم العشوائي في البرنامج السابق و هو العدد 5. و لتبيان الفرق بين دليل الحجرة و بين قيمة هذه الحجرة (أي القيمة المخزنة في الحجرة) لنقارن عبارتي الإلحاق التاليتين على افتراض أن قيمة freqs[5] هي 47 و قيمة freqs[6] هي 62.

Freqs[5]:=freqs[5]+1;

Fregs[5]:=fregs[5+1];

عبارة الإلحاق الأولى ستزيد قيمة fregs[5] بمقدار 1 عما كانت عليه قبل عملية الإلحاق أي ستصبح 48 أما عبارة الإلحاق الثانية ستجعل قيمة freqs[5] تساوي قيمة fregs[6] و التي تساوي إلى 62.

تمثيل النسق representing an array:

عندما نصرح عن متحول من النوع نسق فإن المترجم يحجز حجراً (مساحة تخزينية) من الذاكرة تكفي لتخزين قيم هذا النسق طبعاً وفق نوع المعطيات الأساسي الذي تنتمي إليه حجرات النسق و بالعودة إلى البرنامج السابق
فإن المترجم يقوم بحجز 200 بايت في الذاكرة من اجل النسق السابق و يمكننا التأكد من ذلك إذا أضفنا السطر التالي ضمن البرنامج الرئيسي للبرنامج السابق testDigits كالتالي:

```
Writeln('freqs take ',sizeof(freqs),' bytes of storage');
```

تخزن محتويات حجرات النسق وفق تتابع من حجرات الذاكرة مع تخصيص الحجم الكافي من الذاكرة لكل حجرة من حجرات النسق و اعتماداً على نوع المعطيات الأساسي الخاص بالنسق.
و الشكل المنطقي التالي يبين لنا طريقة تمثيل النسق ضمن الذاكرة:

عنوان الحجرة في الذاكرة	قيم حجرات الذاكرة	دليل حجرات النسق
X	48	0
X+2	44	1
X+4	50	2
X+6	52	3
	....	.....
	....	.....
X+192	43	96
X+194	62	97
X+196	75	98
X+198	78	99

نلاحظ من خلال الشكل السابق أن قيم أدلة النسق لا تخزن في الذاكرة و إنما تخزن قيم حجرات النسقمثل 48 و 44 و 78 ....
أما قيم أدلة النسق فيستخدمها المترجم compiler في حسابه لعناوين مواضع الذاكرة المخصصة لحجر النسق.

قوانين الأنساق rules for arrays:

لا يمكن لعدد حقيقي أن يكون قيمة دليل لنسق.
بعض التعريفات الصحيحة لأنساق مستخدمين أنواع معطيات مختلفة لتمثيل أدلة هذه الأنساق:

```
Type  myordinaltype=(one,two,three,four
,five,six,seven,eight,nine,ten,eleven,twelve,thirteen,fourteen);
Wordarray=array[-5..8] of word; (* integer indexes: array has 14 cell *)
Charindexarray=array['a'..'t'] of real;   (* char indexes:  array has 20 cell *)
Mytypeindexarray=array[one.. fourteen] of char;   (* myordinaltype  indexes:  array has 14 cell *)
Boolindexarray[false..true] of myordinaltype;  (* Boolean indexes: array has 2 cells *)
```

الأنساق كوسطاء arrays as parameters:

في البرنامج السابق testDigits يظهر لنا إمكانية تمرير متحولات الأنساق على أنها متحولات وسيطية arguments للإجراءات و التوابع و التي يمكن أن تمرر بواسطة قيمها passed by value كما هو مبين في الإجراء DispRemainders أو تمرر مرجعياً passed by reference كما هو موضح بالإجراء initremainders.
و في الحقيقية عندما يمرر النسق إلى روتين ما فإن عملية الوصول إلى حجرة هذا النسق تستخدم نفس الدليل و لكن من خلال اسم موضعي آخر للنسق فمثلاً : استخدمنا المميز vals كاسم موضعي للنسق ضمن افجراءين السابقين initremainders و dispremainders .
و الإجراء dispremainders يظهر محتويات النسق واضعاً محتويات كل أربع حجرات على سطر مستقل.
و نلاحظ من خلال الخرج السابق أن قيمة الحجرة freqs[74] (و هي الحجرة الخامسة و الخمسون من حجرات النسق لأننا بدأ دليل الحجر بالرقم صفر) بعد انتهاء تنفيذ البرنامج هي 62 و هذا يعني أن العدد 74 قد تم توليده 62 مرة من خلال 5000 محاولة توليد لأعداد عشوائية صحيحة و بشكل مشابه فإن عدد مرات تكرار العدد 93 هو 48 مرة.

وسطاء الأنساق المفتوحة open array parameters:

زودتنا لغة التربو باسكال بإمكانية استخدام الأنساق المفتوحة كوسطاء ضمن الإجراءات و التوابع فعلى سبيل المثال :رأس التابع التالي يبين لنا طريقة استخدام الأنساق المفتوحة كوسطاء:

Function maxval (var vals:array of integer):integer;

يخبر رأس التابع السابق المترجم بأن نسقاً من الأعداد الصحيحة سيُمرر إليه مرجعياً و لكنه نسق مفتوح أي لم يُحدد حجم النسق و يُحدد حجم النسق هذا عندما يُستدعى التابع السابق مع متحول وسيطي عبارة عن نسق من الأعداد الصحيحة محدد الطول و حتى نتمكن من استخدام الوسطاء المفتوحة يجب أن نستخدم توجيه المترجم التالي (*$P+*) في بداية البرنامج و الذي يخبر البرنامج بأن لديه ستخداماً لوسطاء مفتوحة ضمن البرنامج (ويمكن عدم كتابة هذا التوجيه بالنسخة السابعة لأن المترجم يستطيع مباشرة التعامل مع الأنساق المفتوحة).

و البرنامج التالي يوضح طريقة استخدام النسق المفتوح كوسيط:

Code

```
program test;
const size50=50; size100=100;
type arr50=array[1..size50]of integer;
    arr100=array[1..size100]of integer;
var A50:arr50;
a100:arr100;
index,max50,max100:integer;
function largest(var vals:array of integer):integer;
var temp, index:integer;
begin
temp:=-maxint;
for index:=0 to high(vals)do
if vals[index]>temp then
temp:=vals[index];
writeln(index+1:3,' values checked');
largest:=temp;
end;
begin
randomize;
for index:=1 to size50 do
a50[index]:=random(1000);
for index:=1 to size100 do
a100[index]:=random(20000);
max50:=largest(a50);
max100:=largest(a100);
writeln('largest in a50 = ',max50:5);
writeln('largest in a100 = ',max100:5);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
 50 values checked
100 values checked
largest in a50 =   984
largest in a100 = 19915
```

تلاحظ في البرنامج السابق وجود التابع high الذي يعيد أكبر قيمة لدليل النسق الممر إليه كوسيط أي يعيد القيمة n-1ضمن التابع largest حيث n عدد عناصر النسق و ذلك لأن لغة التربو باسكال تعتبر أن حجرات النسق تأخذ الأدلة من 0 و حتى n-1.

التابع high يمكنه العمل مع أنواع معطيات مرتبة حيث يعيد هذا التابع أعظم قيمة تنتمي لهذا النوع فعند تطبيق التابع السابق على نوع معطيات عدد صحيح integer فإنه يعيد اعظم قيمة و هي maxint يمكن تطبيقه على السلاسل الرمزية حيث يعيد هذا التابع الحجم المصرح عنه لهذه السلسلة.

و يوجد تابع آخر يقابله (له وظيفة متتمة لهذا التابع) وهو التابع low و الذي يعيد اصغر قيمة تنتمي إلى نوع المعطيات الذي ينتمي إليه المتحول الوسيطي الممرر إليه. فعندما يمرر متحول وسيطي من نوع نسق فإن هذا التابع يعيد أصغر قيمة لدليل هذا النسق.

والبرنامج التالي وضح عمل التابعين high و low :

Code

```
program test;

type
matrix=array[1..10] of integer;
matrix1=array[-5..5] of integer;
var mat1:matrix;mat2:matrix1;
procedure proc(var mat:array of integer);
begin
writeln('''begin proc''first');
writeln(mat[5]);
mat[5]:=1024;
writeln('high index= ',high(mat));
writeln('low index= ',low(mat));
writeln('''end proc''last');
end;
procedure format(var mat:array of integer);
var i:integer;
begin
for i:=low(mat) to high(mat) do
mat[i]:=i+1;
end;
procedure print(mat:array of integer);
var i:integer;
begin
for i:=0 to high(mat) do
write(mat[i],',');
writeln;
end;
begin
writeln('''start main program''');
writeln('high mat1index=',high(mat1));
writeln('high mat2index=',high(mat2));
writeln('low mat1index=',low(mat1));
writeln('low mat1index=',low(mat2));
format(mat1);
print(mat1);
proc(mat1);
print(mat1);
```

```
writeln;
format(mat2);
print(mat2);
proc(mat2);
print(mat2);
writeln('''end program''');
readln;
end.
```

خرج البرنامج أعلاه هو:

```
'start main program'
high mat1index=10
high mat2index=5
low mat1index=1
low mat1index=-5
1,2,3,4,5,6,7,8,9,10,
'begin proc'first
6
high index= 9
low index= 0
'end proc'last
1,2,3,4,5,1024,7,8,9,10,

1,2,3,4,5,6,7,8,9,10,11,
'begin proc'first
6
high index= 10
low index= 0
'end proc'last
1,2,3,4,5,1024,7,8,9,10,11,
'end program'
```

دائما في الأنساق المفتوحة بلغة التربو باسكال الدليل يبدأ بشكل افتراضي من الدليل رقم صفر.
ليكن لدينا البرنامج التالي:

Code

```
program test;
var s:string;
s1:string[20];
b:boolean;
k:integer;
begin
writeln(high(s));
writeln(low(s));
writeln(high(s1));
writeln(low(s1));
writeln(high(b));
writeln(low(b));
writeln(high(k));
writeln(low(k));
readln;
end.
```

خرج البرنامج أعلاه هو:

```
255
0
20
0
TRUE
FALSE
32767
-32768
```

العمليات على الأنساق operations an arrays:

إذا قمنا بتعريف نوع معطيات نسق و صرحنا عن متحولين بأنهما ينتميان إلى نوع المعطيات هذا فإننا نستطيع جعل هذين النسقيين متطابقين تماماً بعبارة إلحاق واحدة.

لنفترض أن لدينا متحولين freqs1 و freqs2 و للذان ينتميان إلى نوع المعطيات remainders و المعرف ضمن البرنامج السابق testDigits فإن عبارة الإلحاق التالية تساوي هذين النسقين تماماً:

Freqs2:=freqs1;

و يمكن محاكاة عمل عبارة الإلحاق السابقة من خلال الحلقة التالية :

For index:=0 to 99 do
Freqs2[index]:=freqs1[index];

و بعيداً عن عملية الإلحاق الكلية لعناصر النسق فإنه يمكن القول أن عملية الوصول إلى النسق و معالجته تتم حجرة حجرة و بناء على ذلك فإن العمليات الممكن تطبيقها على حجرات النسق تعتمد على نوع المعطيات الأساسي للنسق.

مثال آخر one more example:

يوضح البرنامج التالي بعض التشابهات بين الأنساق وبين المفهوم الرياضي لمتجهات الأشعة حيث أن متجهة الشعاع vector عبارة عن مجموعة مرتبة من القيم تمثل معلومات عن هذا الشعاع إذ يمكن استخدام متجهة الشعاع لتمثيل نقطة في الفراغ إذ تحتاج كل نقطة إلى ثلاث قيم لتمثيلها في الفراغ ثلاثي الأبعاد (x,y,z).

و البرنامج التالي يحسب و يظهر المسافة بين نقطتين تُدخل إحداثياتهما و يتوقف تنفيذ البرنامج إذا أدخل المستخدم نقطتين لهما إحداثيات متساوية(نفس الأحداثيات).

Code

```
program test;
type
axes=(x,y,z);
vector=array[x..z]of real;
var pointA,pointB:vector;
dist:real;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
procedure getpoint(var pt:vector);
begin
getreal('x-value?',pt[x]);
getreal('y-value?',pt[y]);
getreal('z-value?',pt[z]);
end;
function distance3d(ptA,ptB:vector):real;
var sum:real;
   count:axes;
begin
```

```
    sum:=0.0;
    for count:=x to z do
    sum:=sum+sqr(ptA[count]-ptB[count]);
    distance3d:=sqrt(sum);
end;
begin
writeln('enter point information.!');
writeln('to stop, make both point equal.!');
writeln;
repeat
writeln('point A: ');
getpoint(pointA);
writeln('pointB: ');
getpoint(pointB);
dist:=distance3d(pointA,pointB);
writeln('distance between points is ',dist:10:3);
until dist<=0.0;
readln;
end.
```

خرج البرنامج أعلاه شبيه بالتالي من أجل دخل نفطتين من قبل المستخدم:

```
enter point information.!
to stop, make both point equal.!

point A:
x-value?  1
y-value?  2
z-value?  3
pointB:
x-value?  4
y-value?  5
z-value?  6
distance between points is      5.196
point A:
x-value?  3
y-value?  6
z-value?  9
pointB:
x-value?  8
y-value?  6
z-value?  4
distance between points is      7.071
point A:
x-value?  1
y-value?  2
z-value?  3
pointB:
x-value?  1
y-value?  2
z-value?  3
distance between points is      0.000
```

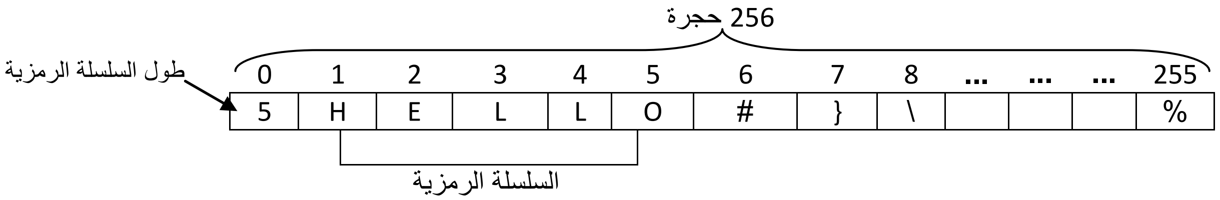
السلاسل الرمزية :strings

مر معنا سابقا أن توع المعطيات string ليس نوع معطيات بسيط و ان تعبيراً مثل التعبير التالي يمثل الرمز السادس ضمن السلسلة الرمزية : mystring[6].

و نوع المعطيات string هو عبارة عن نسق من الرموز و هذا النسق يتضمن و كحد اقصى 255 عنصر إذا لم نحدد الطول الأعظمي لهذا النسق. ويمكن تحدي طول النسق الأعظمي من التصريح التالي الذي يحدد الطول الأعظمي للسلسلة الرمزية وفق 45 رمزاً.

String[45]

و أدلة هذه الأنساق تبدأ بالقيمة 0 وحتى 255 بشكل نظامي حيث تخزن الرموز في الحجرات من 1 و حتى 255 أما القيمة المخزنة في الحجرة 0 فهي تمثل طول هذه السلسلة و طريقة تمثيل السلسلة الرمزية لا تسمح لنا بتخزين أكثر من 255 رمزاً في حين أن النسق يتألف من 256 حجرة و يمكن تمثيل السلسلة الرمزية بالشكل المنطقي التالي:



فإذا انتقلنا للحديث عن عوالم برمجة أكثر جمالا و أناقة يمكنني التحدث عن لغة c أو c++ أو java حيث أن تمثيل الأنساق يتم بشكل مختلف نسبياً فالسلاسل الرمزية ثمثل بأنساق من الرموز تنتي برمز اللاشيء null character وشفرة ASCII لهذا الرمز هي 0 و الإصدار الأخير من تربو باسكال دعمت السلايل المنتهية برمز اللاشيء.

و يوضح البرنامج التالي كيفية استخدام المتحولات من النوع string وكيفية الوصول إلى هذه السلاسل و طريقة العمل مع هذه الرموز حيث يقرأ البرنامج سلسلة رمزية يدخلها المستخدم و يقوم بترتيب الرموز و تخزينها **باستخدام خوارزمية الترتيب التبادلي (exchange sort) أو ما يعرف بالترتيب الفقاعي bubble sort:**

Code

```
program test;
const emptystring='';
var currstr:string;
   count:integer;
procedure getstring(message:string; var info:string);
begin
write(message,' ');
readln(info);
end;
procedure strbubblesort(var info:string);
var temp:char;
   low,high,top,size:integer;
begin
size:=length(info);
top:=size+1;
while(top>1)do
begin
low:=1;
high:=2;
while(high<top)do
begin
    if(info[low]>info[high])then
    begin
        temp:=info[low];
        info[low]:=info[high];
        info[high]:=temp;
    end;
```

```
low:=low+1;
high:=high+1;
end;{end high<top}
top:=top-1;
end;{end top>1}
end;
begin
count:=0;
writeln('enter your strings.');
writeln(' to stop, enter an empty string.');
writeln;
repeat
getstring(': ',currstr);
count:=count+1;
writeln(currstr);
strbubblesort(currstr);
writeln(currstr);
until currstr=emptystring;
writeln(count-1:3,' lines read');
readln;
end.
```

خرج البرنامج أعلاه شبيه بالتالي:

```
enter your strings.
 to stop, enter an empty string.

:  khaled
khaled
adehkl
:  arabic
arabic
aabcir
:  syria
syria
airsy
:  syrian arab republic
syrian arab republic
  aaabbceiilnprrrsuy
:  afdecdefadnb
afdecdefadnb
aabcdddeeffn
:  open
open
enop
:


  6 lines read
```

خوارزمية الفرز الفقاعي the bubble sort algorthim:

تعتبر عبارة statement استدعاء الإجراء strbubblesort نواة البرنامج السابق حيث يرتب هذا الإجراء الأحرف المكونة للسلسلة الرمزية معتمداً على شفرة ASCII للحروف المراد ترتيبها و اعتماداً على حقيقية أن الأحرف لها شفرات ASCII تقابل هذا الترتيب و هذا يعني أن الأسطر التي تبدأ بالحرف A ستكون في بداية النص المرتب أما الأسطر التي تبدأ بالرمز Z فستكون قريبة من نهايته و بما أن الأحرف الكبيرة تأتي ضمن ترتيب حروف ASCII قبل الأحرف الصغيرة فإن الأسطر التي تبدأ بالرمز Z ستمون قبل الأسطر التي تبدأ بالحرف a . و تعتمد خوارزمية الترتيب التبادلي أو الاسم الأشهر لها و هو خوارزمية الفرز الفقاعي على العمليات التالية:

1. تعبر كامل النسق لكل عنصر من عناصر (أحرف) السلسلة الرمزية عدداً من المرات أقل بمرة من العدد الكلي لعناصر النسق.
2. تقارن أثناء كل عبور محتويات حجرتين متجاورتين. فمثلاً في أول عبور ستقارن الخوارزمية محتوى الحجرتين 1 و 2 ومن ثم محتوى الحجرتين 2 و 3 ومن ثم 3 و 4 و هكذا إلى نهاية حجرات النسق (n-1 و n حيث nعدد عناصر النسق)
3. و أثناء عملية المقارنة تنجز عملية تبديل بين محتويات الحجرات (إذا تطلب الأمر ذلك) لكي تحتوي الحجرة ذات الرقم اللأكبر على القيمة الأكبر.
4. بعد أن تنتهي مقارنة محتويات جميع الحجرات فإن آخر حجرة من هذه الحجرات أي الحجرة العليا سوفتحتوي العنصر المرتب الأخير .
5. وبما أن الحجرة الأخيرة من السلسلة الرمزية قد وضعت فيها القيمة الصحيحة المرتبة إذاً عند عملية العبور الثانية ستحتوي الحجرة قبل الأخيرة على العنصر المرتب المناسب و هكذا حتى نهاية عبور السلسلة الرمزية و بذلك تغدو عناصر السلسلة الرمزية مرتبة بعد انتهاء عمليات العبور.

ويمكن ترتيب السلسلة بشكل تنازلي.

الأنساق متعددة الأبعاد multidimensional arrays:

يمكن أن يكون نوع المعطيات الأساسي الخاص بالنسق أي نوع من أنواع المعطيات بما في ذلك نوع المعطيات نسق فمثلاً:إذا أردنا أعلى تسجيل درجة حرارة أثناء النهار و اعلى درجة حرارة أثناء اليل يومياً و طيلة شهر فإن إنجاز هذا العمل يدوياً يحتاج إلى جدول شبيه بالجدول التالي:

Date	Day	Night
1		
2		
3		
4		
…		
…		
…		
29		
30		
31		

يتضمن الجدول السابق 62 حجرة مختلفة متجمعة ضمن 31 زوجاً من الحجرات حيث يمثل كل سطر من هذا الجدول زوجاً من القيم أما العمود date فيمثل دليلاً لهذه القيم و ليس جزءاً من المعلومات. ويمكن اعتبار هذه القيم على أنها 62 حجرة مستقلة أو على أنها 31 حجرة كل منها تحتوي على حجرتين و يمكن بناء مثل هذه الأنساق بطرق متعددة و منها:

```
Type     dailytemps=array[1..2] of real;
         Tempdata1=array[1..31] of dailytemps;
         Tempdata2=array[1..31] of array [1..2] of real;
         Tempdata3=array[1..31,1..2] of real;
```

يتألف النسق الأول tempdata1 من 31 حجرة من النوع dailytemp ونوع المعطيات هذا معرف على أنه نسق يتألف من حجرتين من النوع عدد حقيقي real و يتألف النسق الثاني tempdata2 أيضاً من 31 حجرة كل حجرة منها عبارة عن نسق يتألف من حجرتين من النوع عدد حقيقي real ولكن استخدمنا هنا أسلوباً آخر في التصريح عن هذا النسق المطابق للنسق tempdata1 أما النسق الثالث tempdata3 فيتألف من نسق ثنائي الأبعاد ويتألف بُعده الأول من 31 عنصر(سطر) و بُعده الثانس يتألف من عنصرين (عمودين).

الوصول إلى العناصر في الأنساق متعددة الأبعاد accessing elements in multidimensional array:

يمثل الجدول السابق مصفوفة matrix رياضية تتألف من أسطر من القيم و كل سطر يمثل متجهة شعاع vector تملك عدداً من القيم و عدد القيم هذه متساو في كل سطر وأول قيمة في متجهة الشعاع هي القيمة الموجودة في أول عمود من أعمدة المصفوفة و هكذا. و بالتالي فإن الجدول السابق يمثل مصفوفة عدد أسطرها 31 سطر و عدد أعمدتها عمودين و عملية تشبيه النسق بالمصفوفة يساعدنا كثيراً في التعامل مع الأنساق و في الوصول إلى حجرات هذه الأنساق.
و البرنامج التالي يبين لنا كيفية الوصول إلى حجرات النسق:

Code

```
program test;
type
matrix=array[1..5,1..3] of integer;
var samplematrix:matrix;
procedure fillmatrix(var vals:matrix);
var row,column:integer;
begin
for row:=1 to 5 do
for column:=1 to 3 do
vals[row,column]:=10*row+column;
end;
procedure dispmatrix(vals:matrix);
var row,column:integer;
begin
writeln('col 1':13, 'col 2':8 ,'col 3':8);
for row:=1 to 5 do
begin
write('row ',row);
for column:=1 to 3 do
write(vals[row][column]:8);
writeln;
end;
end;
begin
fillmatrix(samplematrix);
dispmatrix(samplematrix);
readln;
end.
```

خرج البرنامج السابق هو:

```
        col 1   col 2   col 3
row 1      11      12      13
row 2      21      22      23
row 3      31      32      33
row 4      41      42      43
row 5      51      52      53
```

أنساق السلاسل الرمزية string arrays :

الأنساق المشكلة من سلاسل رمزية هي مثال آخر عن الأنساق ثنائية البعد و يظهر لنا البرنامج التالي مثالاً عن مثل هذه الأنساق:

Code

```
program test;
const maxmenu=20;
type strarray=array[1..maxmenu] of string;
var menu:strarray;
procedure initmenu(var menu:strarray);
var index:integer;
begin
menu[1]:='choice1';
menu[2]:='choice2';
menu[3]:='choice3';
for index:=4 to maxmenu do
menu[index]:='';
end;
procedure showmenu(var menu:strarray;nrchoices:integer);
var index:integer;
begin
if nrchoices>maxmenu then
nrchoices:=maxmenu;
for index:=1 to nrchoices do
writeln(menu[index]);
writeln;
write('your choices? (1 to ',nrchoices, ') ');
end;
begin
initmenu(menu);
showmenu(menu,3);
readln;
end.
```

خرج البرنامج السابق هو:

```
choice1
choice2
choice3

your choices? (1 to 3)
```

يحتوي البرنامج السابق على إجراءين و هذان الإجراءين يظهران قائمة من الخيارات من أجل التخاطب مع المستخدم و مما تجب ملاحظته أن البعد الأول و الذي يمثل الأسطر استخدم فقط عند معالجة عناصر النسق menu وذلك لأن السلاسل الرمزية تُعامل على أنها متحول بسيط يتألف من تتابع معين من الرموز أي أننا اعتبرنا أن هذا النسق ذا بعد واحد و لكن يمكن استخدام هذا النسق على أنه ذو بعدين و ذلك من أجل الوصول إلى الرموز المكونة للسلاسل الرمزية بشكل مستقل و ذلك بإضافة دليل ثانٍ يمثل رموز السلسلة فعلى سبيل المثال: يمكن أن نشير إلى الرمز السادس في السلسلة الرمزية الثالثة و هو 'e' كالتالي:

Menu[3,6]

و الأنساق تعامل كباقي أنواع المعطيات عند تمريرها كوسطاء فإذا مررنا النسق بواسطة قيمته فإن نسخة كاملة من النسق سوف تمرر إلى البيئة الموضعية (المحلية) للروتين المستدعي فعلى سبيل المثال من أجل نسق من menu فإن نسخة من هذا النسق سوف تحتاج إلى 5120 بايت من الذاكرة و فقدان مثل هذه الكمية من الذاكرة و التي قد تكون أكثر من ذلك مع البرامج الكبيرة قد

يسبب إرباكاً في استخدام الذاكرة (طبعاً البرامج الضخمة جداً) . أما إذا مررنا النسق menu مرجعياً لن نحتاج إلى نسخة من هذا النسق لتمريرها إلى البيئة الموضعية للروتين المستدعى لأن الروتين يفترض أن هذا النسق جزء من البيئة المخصصة للمتحولات الوسيطية و التي تحفز عند استدعاء هذا الروتين و هذه الطريقة تختصر زمن الروتين و تقلل من حجرات الذاكرة المستخدمة.

ومما يجب ذكره هنا أن النسخ التي يتم توليدها ضمن البيئة المحلية تحرر عند الانتهاء من تنفيذ الإجراء (طبعا في حال التمرير بالقيمة)

و أخيراً و ليس آخراً يجب أن لا نحاول الوصول إلى عناصر النسق خارج حدود النسق لأن محاولة الوصول تلك سوف تؤدي إلى إنهاء البرنامج مع خطأ في تنفيذه.

الأنساق ثلاثية الأبعاد thre dimensional arrays:

يُظهر البرنامج التالي لنا كيفية تعريف وتصريح و استخدام الأنساق ثلاثية الأبعاد:

Code

```
program test;
type hypermatrix=array[1..3,1..4,1..5]of integer;
var sample:hypermatrix;
procedure fillhypermatrix(var vals:hypermatrix);
var plane,row,column:integer;
begin
for plane:=1 to 3 do
for row:=1 to 4 do
for column:=1 to 5 do
vals[plane,row,column]:=100*plane+10*row+column;
end;
procedure sum(var x,y:integer);
begin
writeln(x+y);
end;
procedure disphypermatrix(vals:hypermatrix);
var plane, row, column:integer;
begin
for plane:=1 to 3 do
begin
writeln('plane ':10,plane);
writeln('col 1':13, 'col 2':8, 'col 3':8, 'col 4':8,
' col 5':8,);
for row:=1 to 4 do
begin
write('row ',row);
for column:=1 to 5 do
write(vals[plane][row][column]:8);
writeln;
end;
end;
end;
procedure shufflehypermatrix(var vals:hypermatrix);
var plane:integer;
begin
for plane:=1 to 3 do
vals[plane]:=vals[3-plane+1];
end;
begin
```

```
fillhypermatrix(sample);
disphypermatrix(sample);
writeln;
shufflehypermatrix(sample);
disphypermatrix(sample);
readln;
end.
readln;
end.
```

خرج البرنامج السابق هو :

```
     plane 1
          col 1     col 2     col 3     col 4     col 5
row 1       111       112       113       114       115
row 2       121       122       123       124       125
row 3       131       132       133       134       135
row 4       141       142       143       144       145
     plane 2
          col 1     col 2     col 3     col 4     col 5
row 1       211       212       213       214       215
row 2       221       222       223       224       225
row 3       231       232       233       234       235
row 4       241       242       243       244       245
     plane 3
          col 1     col 2     col 3     col 4     col 5
row 1       311       312       313       314       315
row 2       321       322       323       324       325
row 3       331       332       333       334       335
row 4       341       342       343       344       345

     plane 1
          col 1     col 2     col 3     col 4     col 5
row 1       311       312       313       314       315
row 2       321       322       323       324       325
row 3       331       332       333       334       335
row 4       341       342       343       344       345
     plane 2
          col 1     col 2     col 3     col 4     col 5
row 1       211       212       213       214       215
row 2       221       222       223       224       225
row 3       231       232       233       234       235
row 4       241       242       243       244       245
     plane 3
          col 1     col 2     col 3     col 4     col 5
row 1       311       312       313       314       315
row 2       321       322       323       324       325
row 3       331       332       333       334       335
row 4       341       342       343       344       345
```

يعمل البرنامج السابق مع أنساق ثلاثية الأبعاد فالإجراء fillhypermatrix يملأ حجرات النسق بأعداد صحيحة أما الإجراء disphypermatrix فيظهر هذه القيم واما الإجراء shufflehyper فيظهر كيفية إلحاق مصفوفة كاملة من هذا النسق الذي يتألف من مصفوفات و ذلك بإهمال أدلة الأسطر و الأعمدة.
أما إذا أردنا الوصول إلى سطر كامل من هذا النسق فيجب إهمال دليل الأعمدة فقط.
ومما يجب ذكره هنا أن الأنساق يمكن ان تكون من أبعاد مختلفة (رباعية البعد – خماسية البعد.....)

نقاط متفرقة حول الأنساق arrays miscellaneous points:

1. **نوع المعطيات مجال جزئي the subrange type:**

تملك لغة باسكال نوع معطيات بسيط يبنى من نوع معطيات بسيط آخر و يمثل حدوداً من القيم التي تنتمي إلى نوع المعطيات الأصلي .

يعرف نوع المعطيات هذا و يصرح عن متحولات تنتمي إليه قبل استخدامه لأن المترجم يحتاج إلى أن معرفة مجال القيم التي تنتمي إلى نوع المعطيات هذا.فعلى سبيل المثال لنفترض أننا نريد استخدام الأرقام من 0 إلى 9 لذلك نعرف نوع معطيات يمثل هذه القيم و نصرح عن متحولات تنتمي إليه كالتالي:

```
Type singledigit=0..9;
Var   digitval :singledigit;
```

تخبر العبارة الأولى المترجم بوجود نوع معطيات اسمه singledigit و أن المتحول الذي ينتمي إلى هذا النوع يمكن ان يأخذ قيمة من 0 و حتى 9 ومما تجب ملاحظته هنا إلى أننا لم نستخدم أقواس حول المجال الجزئي لأن الأقواس تكون ضرورية فقط عند إخبار المترجم compiler بأن لديه قيماً جديدة كما نفعل عندما نعرف قيماً تعدادية و في المثال السابق كان المترجم يعرف مسبقاً الأعداد الصحيحة لذلك من السهل عليه تفسير التركيب 0..9 و نوع المعطيات مجال جزئي يستخدم بكثرة لتحديد أدلة الأنساق كما موضح في التعريفات التالية:

```
Type     singledigit=0 .. 9;
         Facevalue=2..14;
         Cardsuit=(club,diamond,heart, spade);
         Redsuit=diamond.. heart ;
         Smallval=array[singledigit]of real;
         Redcards=array [facevalue,redsuit] of Boolean;
         Anycards=array [facevalue,cardsuit] of Boolean;
Var      digitdata:smallval;
         Redhand: redcards;
         Hand: anycards;
```

إذا فحصنا التعريفات السابقة نلاحظ أن تعريف المجال الجزئي قد سبق كل التعريفات لكي يفهم المترجم بأن المميز singledigit هو البديل المكافئ للتركيب 0..9 . و لذلك عندما استخدمنا هذا المميز ضمن المجال المحدد للنسق و بين قوسين مربعين استبدل المترجم و ببساطة هذا المميز بمكافئه . أي أننا عرفنا نسقاً يحتوي عشرة عناصر وفق مجال لدليل النسق من 0 وحتى 9.

أما تعريف النسق الثاني فقد استخدم تعريف النوع التعدادي cardsuit الذي حدد أن المتحول الذي ينتمي لهذا النوع يمكن أن يأخذ إحدى القيم التاية: club أو diamond أو heart أو spade .

و بعد تعريف هذا النوع عرفنا مجالاً جزئياً يعتمد هذا النوع اساساً له و هذا المجال هو redsuit و ذلك بذكر القيمة الابتدائية و القيمة النهائية لهذا المجال واضعين بينهما نقطتين متتاليتين .. و بدون إحاطتهم بأقواس في حين أننا أحطنا نوع المعطيات التعدادي بأقواس لأن المترجم لا يعرف هذه القيم مسبقاً و استخدم تعريف النسق الثاني مجالاً جزئياً آخر هو facevalue و يأخذ قيماً من 2 و حتى 14 كبعد أول و استخدم المجال الجزئي redsuit الذي يأخذ قيمتين فقط كبعد ثاني أي عدد حجرات النسق الثاني هي 2*13=26 وم نثم صرح عن المتحول redhand على أنه نسق من النوع redcards و يمكن الوصول إلى لحجرة الأولى في هذا النسق من خلال التركيب redhand[2][diamond] أما الحجرة الثانية فيمكن الوصول إليهما بواسطة التركيب redhand[2][heart] أما الحجرتان قبل الأخيرة و الأخيرة فيمكن الوصول إليهما بواسطة التركيبين التاليين و على الترتيب: readhand[14][diamond] و redhand[14][heart] .

أما تعريف النسق الثالث anycards فهو مشابه لتعريف النسق redcards إلا أن النسق anycards استخدم المجال الكال للنوع التعدادي cardsuit في بُعده الثاني و هذا الاستخدام مسموح لأن النوع التعدادي في حقيقته مجال جزئي.

2. **الأنساق الثابتة array constants :**

يمكن تعريف الأنساق الثابتة في لغة التربو باسكال على أنها ثوابت تحدد قيمتها أثناء تعريفها و هذه القيم لن تكون قابلة للتغيير طوال فترة تنفيذ البرنامج.

و البرنامج التالي يبين لنا طريقتين في تعريف أنساق ثابتة:

Code

```
program test;
const powersofthree:array[1..6] of integer=(1,3,9,27,81,243);
type testarray=array [1..5] of real;
const sqrtarray:testarray=(1,1.414,1.732,2,2.236);
var count:integer;
```

```
begin
for count:=1 to 6 do
writeln(powersofthree[count]);
for count:=1 to 5 do
writeln(sqrtarray[count]:6:2);
readln;
end.
```

يرينا تعريف الثابت الأول powersofthree كيفية تعريف النسق كثابت و تهيئته ضمن خطوة واحدة حيث أننا عرفنا هذا النسق كثابت وفق العبارة:

Const powersofthree:array [1..6] of integer= (1,3,9,27,27,81,243);

وبعدها هيأنا هذا النسق (أي أعطينا قيماً لحجرات هذا النسق) بإضافة إشارة المساواة في آخر هذا التعريف و من ثم وضعنا القيم المخصصة لهذا النسق و هي عبارة عن ست قيم صحيحة مفصولة عن بعضها بفاصلة (,) و موضوعة ضمن قوسين.
و يمكن الوصول إلى حجرات هذا النسق تماما كما هو الحال بالنسبة للأنساق العادية.
أما تعريف النسق الثاني sqrtarray فهو يختلف عن سابقه و يقدم لنا طريقة أخرى في التصريح عن الأنساق الثابتة حيث أننا عرفنا النسق testarray بشكل طبيعي على أنه نسق من الأعداد الحقيقية real و لكن التصريح عن النسق الثابت تم في السطر الذي يليه حيث ذكرنا اسم النسق السابق testarray و الذي أصبح معروفاً بالنسبة للمترجم و من ثم أعطينا قيماً لهذا النسق بنفس الطريقة التي استخدمناها في تهيئة النسق الأول.
ونرود الان تعريفين لنسقين ثابتين من النوع رمز char:

Code

```
program test;
type alpharray = array['a'..'z'] of char;
const alfabet:alpharray=('3','b','c','d','e','f','g','h',
                    'i','j','k','l','m','n','o','p',
                    'q','r','s','t','u','v','w','x'
                    ,'y','z');
alphabet:alpharray='abcdefghijklmnopqrstuvwxyz';
begin

writeln(alphabet);
writeln(alfabet);
writeln(alphabet['a']);
writeln(alfabet['a']);
readln;
end.
```

خرج البرنامج السابق هو:

```
abcdefghijklmnopqrstuvwxyz
3bcdefghijklmnopqrstuvwxyz
a
3
```

تعريف النسق الثابت الأول alphabet يهيئ النسق بواسطة تحديد الرموز المخصصة لحجرات النسق رمزاً رمزاً أما تعريف النسق الثاني alphabet فيهيء النسق بطريقة تعتمد على السلاسل الرمزية لأن التهيئة تمت بإلحاق سلسلة رمزية من 26 رمزاً بالنسق الثابت.

3. الأنساق متعددة الأبعاد multidimensional :
يمكن أيضاً تعريف أنساق ثابتة متعددة الأبعاد و تجري تهيئة هذه الأنساق بوضع قيمة كل حجرة ضمن قوسين و كمثال على هذه الأنساق نورد التعريفين التاليين:

Code

```
program test;
type twod=array [1..3,1..2] of integer;
   threed=array[1..3,1..2,1..2] of integer;
const twins:twod=((11,12),(21,22),(31,32));
    triplets:threed=(  ((111,112),(121,122)),
                  ((131,132),(211,212)),
                  ((221,222),(231,232)) );
var plane,row,column:integer;
begin
writeln('col1':13,'col2':8);
for row:=1 to 3 do
begin
write('row',row);
for column:=1 to 2 do
write(twins[row][column]:8);
writeln;
end;
writeln('---------------------------');
for plane:=1 to 3 do
begin
writeln('plane ':10,plane);
writeln('col1':13, 'col2':8);
for row:=1 to 2 do
begin
write('row',row);
for column:=1 to 2 do
write(triplets[plane][row][column]:8);
writeln;
end;
end;
readln;
end.
```

خرج البرنامج السابق هو:

```
         col1    col2
row1       11      12
row2       21      22
row3       31      32
---------------------------
    plane 1
         col1    col2
row1      111     112
row2      121     122
    plane 2
         col1    col2
row1      131     132
row2      211     212
    plane 3
         col1    col2
row1      221     222
row2      231     232
```

الشكل المنطقي التالي يبين محتويات النسق الثابت twins:

	Col1	Col2
Row1	11	12
Row2	21	22
Row3	31	32

و الأشكال التالية تبين محتويات النسق الثابت triplets:

Plane1

	Col1	Col2
Row1	111	112
Row2	121	122

Plane2

	Col1	Col2
Row1	131	132
Row2	211	212

Plane3

	Col1	Col2
Row1	221	222
Row2	231	232

نوع المعطيات سجل the record type:

قد نحتاج في بعض الأحيان إلى ان نجمع بعضاً من المعلومات التي تنتمي إلى نوع معطيات واحد. فعلى سبيل المثال لنفترض أننا نريد تمثيل معلومات حول بلد ما فإن هذه المعلومات سوف تتضمن اسم هذا البلد و عاصمته و عدد السكان و مساحة هذا البلد و هذه المعلومات كما نلاحظ تنتمي إلى أنواع معطيات مختلفة حيث ينتمي اسم البلد و عاصمته إلى نوع المعطيات سلسلة رمزية string أما عدد السكان و مساحة البلد فتنتمي إلى نع المعطيات عدد صحيح integer.

و يستطيع السجل record التعامل مع هذه المتحولات إذ يمثل معلومات يمكن أن تحتوي على أنواع معطيات مختلفة و بما أن نوع المعطيات سجل هو نوع مركب إذاً يجب تعريفه في القسم type قبل استخدامه و فيما يلي مثال على نوع المعطيات هذا :

```
Type country=record
          Name,capital: string[80];
          Pop,area:longint;
     End;
```

يزودنا التعريف السابق بنوع معطيات سجلفيه أربعة عناصر : عنصران منهم عبارة عن سلسلة رمزية بطول 80 رمزاً و العنصران الآخران أعداد صحيحة طويلة longint و التصريح عن العناصر أتى بين المميزين المحجوزين record و ;end و عناصر السجل تُعرف كحقول fields للسجل و الصيغة الكتابية لتعريف السجل هي:

<مميز>=record < لائحة من الحقول> end;

و تعريف هو أحد المواضع الثلاثة في لغة باسكال والتي نضع المميز end بدون المميز begin و الموضعين الآخرين هما عبارة case و نوع المعطيات كائن.

الوصول إلى الحقول ضمن السجل accessing fields in a record:

يبين البرنامج التالي طريقة الوصول إلى حقول السجل country:

Code
program test; type country=record

```
        name,capital:string[80];
        pop,area:longint;
        end;
var sample:country;
procedure initcountry(var thecountry:country; cname,capital:string;
                                  population,size:longint);
begin
thecountry.name:=cname;
thecountry.capital:=capital;
thecountry.pop:=population;
thecountry.area:=size;
end;
procedure dispcountry(thecountry:country);

begin
write('name     : ',thecountry.name:20);
writeln(';  capital: ',thecountry.capital:17);
write('population: ',thecountry.pop:10,' ':10);
writeln(';  area   :',thecountry.area:10,' sq.mi.');
end;
begin
initcountry(sample,'syriaarabic','damascus',
23000000,185000);
dispcountry(sample);
readln;
end.
```

خرج البرنامج السابق هو:

```
name      :          syriaarabic;  capital:          damascus
population:    23000000           ;  area    :     185000 sq.mi.
```

يعرف البرنامج السابق نوع المعطيات country و يصرح عن المتحول sample الذي ينتمي لنوع المعطيات هذا.
و يُظهر الإجراءان initcountry و dispcountry كيفية تمرير السجلات لوسطاء إلى الروتينات إما مرجعياً أو بقيمتها إذ إن عملية تمرير المتحولات من النوع سجل تشبه عملية تمرير متحولات من أنواع المعطيات الأخرى و يتم الوصول إلى حقول السجلات وفق ما يلي:

<اسم المتحول من النوع سجل>. <اسم الحقل>

وتعريف متحولات السجل مشابه تماما لتعريف متحولات الأخرى :

<مميز>:< نوع المعطيات>

يمكننا ترك الفراغات بين الأسماء و النقطة كما نريد و بالتالي العبارات الأربع التالية متكافئة تماماً:

```
Thecountry.area  :=size;
Thecountry   .   area  :=size;
Thecountry.   area  :=size;
Thecountry   .area  :=size;
```

عندما يتدعى الإجراء dispcountry فإن محتويات السجل sample تنسخ و ت}خذ هذه النسخة و تخزن على أنها محتويات للسجل thecountry ومن ثم يُظهر الإجراء محتويات هذا السجل كما في الخرج السابق.

المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com
Syrian Arab Republic
دمشق-معضمية الشام

و تخزن محتويات السجل sample في الذاكرة على الشكل التالي:

اسم الحقل	محتويات الحقل	العنوان في الذاكرة
Name	syriaarabic	X
Capital	damascus	X+81
Pop	23000000	X+162
Area	185000	X+166

أنواع المعطيات المركبة كحقول ضمن نوع المعطيات سجل structured types as record fields:

يملك نوع المعطيات سجل السابق country حقلين من النوع سلسلة رمزية string و هما عبارة عن نسقين من الرموز أي يمكننا استخدام انواع المعطيات المركبة كحقول ضمن سجل ما.

و في الحقيقة يمكننا استخدام أي نوع معطيات ضمن حقول السجل ما عدا نوع المعطيات ملف file و نفس نوع المعطيات سجل الذي نعرفه حالياً أي لا نستطيع استخدام حقول من النوع country ضمن السجل country.

ونستطيع استخدام انساق من الأعداد الصحيحة أو الحقيقية او أنساق من السلاسل الرمزية كحقول أو سجلات أخرى كحقول ضمن سجل ما و تعطينا التعريفات التالية أمثلة عن كيفية تعريف نوع المعطيات سجل و كيفية التصريح عن متحولات تنتمي لأنواع المعطيات المعرفة تلك:

Code

```
program test;
type freqarray=array[0..10] of integer;
cointrials= record
          freqs:freqarray;
          probability:real;
          end;
menu=array[1..20] of integer;
menurec=record
       choices:menu;
       showextendedmenu:boolean;
       end;
miscinfo=record
        money:string[80];
        density, literacy:real;
        end;
country=record
       name,capital:string[80];
       pop,area:longint;
       end;
nation=record
       land:country;
       data:miscinfo;
       end;
var flips:cointrials;
   currmenu:menurec;
   samplenation:nation;
   begin
   readln;
   end.
```

يملك السجل contrails السابق حقلين أحدهما عبارة عن نسق من النوع freqarray و الذي يحتوي على 11 عدداً صحيحاً و الحقل الآخر عبارة عن عدد حقيقي و عملية الوصول إلى إلى حقول هذا السجل سوف يوضحها الإجراء التالي:

Code

```
procedure initcointrials(var vals:cointrials);
var index:integer;
begin
for index:=0 to 10 do
vals.freqs[index]:=0;
end;
begin
readln;
end.
```

و يملك السجل menurec على حقلين أحدهما عبارة عن نسق ثنائي الأبعاد (بسبب كون نوع المعطيات التي ينتمي إليه النسق هو سلسلة رمزية و التي لا تخرج عن كونها نسقاً أحادياً) و الآخر عبارة عن قيمة بوليانية و عملية الوصول إلى الحقل الأول في هذا السجل تخضع لاجتماع قانون الوصول إلى حقول السجل وقانون الوصول غلى حجرات النسق ثنائي البعد فحتى نصل إلى رمز معين ضمن النسق علينا أولاً الوصول إلى النسق و من ثم تحديد الرمز المطلوب من خلال أدلة النسق كما هو موضح في المثال التالي الذي يحدد الرمز الرابع ضمن السلسلة الثالثة ضمن الحقل choices للسجل currmenu :

Currmenu.choices[3][4]

و للوصول إلى حقول سجل ما يجب علينا أولاً ذكر اسم السجل متبوعاً بنقطة و من ثم نتعامل مع الحقل وفق نوع المعطيات الذي ينتمي إليه و في حال كون هذا الحقل ينتمي إلى نوع معطيات مركب فينبغي التعامل معه وفق طرق الوصول إلى عناصر نوع المعطيات المركب هذا و البرنامج التالي يعطينا مثالاً عن سجل حقوله تنتمي إلى أنواع معطيات مركبة و يوضح البرنامج كيفية الوصول إلى حقول هذا السجل :

Code

```
program test;
type
miscinfo=record
       money:string[80];
       density, literacy:real;
       end;
country=record
      name,capital:string[80];
      pop,area:longint;
      end;
nation=record
      land:country;
      data:miscinfo;
      end;
var
   samplenation:nation;
procedure initcountry(var thecountry:country; cname,capital:string;
                              population,size:longint);
begin
thecountry.name:=cname;
thecountry.capital:=capital;
thecountry.pop:=population;
thecountry.area:=size;
end;
procedure dispcountry(thecountry:country);
```

```
begin
write('name     : ',thecountry.name:20);
writeln(';  capital: ',thecountry.capital:17);
write('population: ',thecountry.pop:10,' ':10);
writeln(';  area   :',thecountry.area:10,' sq.mi.');
end;
procedure dispnation(thenation:nation);
begin
dispcountry(thenation.land);
writeln('currency  = ',thenation.data.money);
writeln('density   = ',thenation.data.density:10:3);
writeln('literacy  = ',thenation.data.literacy:10:3,'%');
end;
begin
initcountry(samplenation.land,'syria arabic','damascus',
                        23000000,185000);
samplenation.data.money:='lira';
samplenation.data.density:=45.5;
samplenation.data.literacy:=98.9;
dispnation(samplenation);
readln;
end.
```

خرج البرنامج السابق هو:

```
name      :          syria arabic;  capital:          damascus
population:    23000000           ;  area    :    185000 sq.mi.
currency   = lira
density    =     45.500
literacy   =     98.900%
```

يرينا استدعاء الإجراء initcountry كيفية تمرير سجل ما لوسيط حتى ولو كان هذا السجل حقلاً في سجل آخر فقد مررنا السجل land الموجود ضمن السجل samplenation كمتحول وسيطي argument عند استدعاء الإجراء initcountry و من خلال عملية التمرير السابقة نلاحظ عدم وجود فرق في تمرير سجلات كمتحولات وسيطية عن تمرير أي متحول من المتحولات سواء كانت سجلات مستقلة أو سجلات ضمن أنواع معطيات مركبة اخرى حيث يستبدل اسم المتحول الوسيطي بالنتحول الموضعي لهذا السجل ضمن الروتين فمثلاً: استبدل اسم السجل الذي مرر كمتحل وسيطي samplenation باسم المتحول الموضعي local variable المقابل له و هو thecountry ضمن الإجراء initcountry (طبعاً عندما استدعي هذا الإجراء)

و البرنامج السابق يبين لنا كيفية الوصول إلى حقل ضمن حقل و ذلك بالوصول أولاً إلى الحقل الخارجي ومن ثم الحقل الذي في داخله و هكذا. فمثلاً : حتى نصل إلى الحقل literacy علينا أولاً الوصول إلى الحقل data للسجل samplenation ومن ثم الوصول إلى الحقل literacy الموجود ضمن السجل data كالتالي:

Samplenation.data.literacy;

استخدام عبارة with لتقصير(اختصار) أسماء الحقول using with to shorten field names:

قد نستخدم سجلاً ما كحقل ضمن سجل آخر ووفق مستويات متعددة و من أجل الوصول إلى حقل ما ضمن مستوى معين علينا ببساطة ذكر أسماء السجلات حتى نصل إلى الحقل المطلوب إذ يجب ان يكون المميز الذي يقع على يسار النقطة اسماً للسجل و المميز الذي يقع على يمين النقطة يجب أن يكون حقلاً فيه. فإذا كان لدينا عدد كبير من السجلات المتداخلة مع بعضها فسوف نحصل على عدد كبير من النقاط كل منها تشير إلى السجل التالي و بذلك تصبح أسماء الحقول طويلة جداً كما هو موضح في البرنامج التالي:

إعداد المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com

Code

```
program test;
type
shortstr=string[30];
education=record
        major,degree:shortstr;
        GPA:real;
        end;
student=record
      schooling:education;
      Id:shortstr;
      classof:integer;
      end;
graduate=record
       firstname,lastname:shortstr;
       age:integer;
       learning:student;
       end;
var grad:graduate;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
procedure getshortstr(message:string; var value:shortstr);
begin
write(message,' ');
readln(value);
end;
procedure getreal(message:string;var value:real);
begin
write(message,' ');
readln(value);
end;
procedure getgraduateinfo(var info:graduate);
begin
getshortstr('first name?',info.firstname);
getshortstr('last name?',info.lastname);
getinteger('Age?',info.age);
getshortstr('ID?',info.learning.id);
getinteger('class of?', info.learning.classof);
getshortstr('Major?',info.learning.schooling.major);
getshortstr('degree?',info.learning.schooling.degree);
getreal('GAP?',info.learning.schooling.GPA);
end;
procedure dispgraduateinfo(info:graduate);
begin
writeln('first name: ',info.firstname);
writeln('last name : ',info.lastname);
writeln('Age       : ',info.age);
```


المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com
Syrian Arab Republic
دمشق-معضمية الشام

```
writeln('ID       : ',info.learning.id);
writeln('class of  : ',info.learning.classof);
writeln('Major     : ',info.learning.schooling.major);
writeln('Degree    : ',info.learning.schooling.degree);
writeln('GAP       : ',info.learning.schooling.GPA:5:2);
end;
begin
getgraduateinfo(grad);
dispgraduateinfo(grad);
readln;
end.
```

يستخدم السجلان education و student المعرفان ضمن البرنامج السابق كأنواع معطيات لحقول ضمن سجلات أخرى فمثلاً الحقل schooling ضمن السجل student ينتمي لنوع المعطيات المعرف قبله و هو education و الحقل learning ضمن السجل graduate ينتمي لنوع المعطيات student ومن خلال هذا التداخل بين السجلات تغدو كتابة اسم حقل ما أمراً مضجراً و طويلاً و الإجران قد جرى تعديلهما عن نسختهما الأصلية ضمن البرنامج السابق و استخدمت فيهما عبارى with :

Code

```
program test;
type
shortstr=string[30];
education=record
        major,degree:shortstr;
        GPA:real;
        end;
student=record
       schooling:education;
       Id:shortstr;
       classof:integer;
       end;
graduate=record
        firstname,lastname:shortstr;
        age:integer;
        learning:student;
        end;
var grad:graduate;
procedure getinteger(message:string; var value:integer);
begin
write(message,' ');
readln(value);
end;
procedure getshortstr(message:string; var value:shortstr);
begin
write(message,' ');
readln(value);
end;
procedure getreal(message:string;var value:real);
begin
write(message,' ');
readln(value);
end;
```

```
procedure getgraduateinfo(var info:graduate);
begin
with info do
begin
getshortstr('first name?',firstname);
getshortstr('last name?',lastname);
getinteger('Age?',age);
end;
with info.learning do
begin
getshortstr('ID?',id);
getinteger('class of?', classof);
end;
with info.learning.schooling do
begin
getshortstr('Major?',major);
getshortstr('degree?',degree);
getreal('GAP?',GPA);
end;
end;
procedure dispgraduateinfo(info:graduate);
begin
with info do
begin
writeln('first name: ',firstname);
writeln('last name : ',lastname);
writeln('Age       : ',age);
end;
with info.learning do
begin
writeln('ID        : ',id);
writeln('class of  : ',classof);
end;
with info.learning.schooling do
begin
writeln('Major     : ',major);
writeln('Degree    : ',degree);
writeln('GAP       : ',GPA:5:2);
end;
end;
begin
getgraduateinfo(grad);
dispgraduateinfo(grad);
readln;
end.
```

و الصيغة الكتابية لعبارة with هي :

With اسم السجل do <عبارة بسيطة أو مركبة>

أنساق السجلات arrays of records:

البرنامج التالي يحتوي على تعريف لنسق من السجلات و هذا البرنامج يُرينا كيفية الوصول إلى حجرات هذا النسق و حجرات حقول هذه السجلات:

Code

```
program test;
const maxval=26;
     maxtrials=1000;
type  entry=record
     freq,max,min:integer;
     end;

intarray=array[0..maxval] of entry;
var  data:intarray;
    rawval,result,count:integer;
procedure initintarray(var ints:intarray);
var index:integer;
begin
for index:=0 to maxval do
begin
ints[index].freq:=0;
ints[index].max:=0;
ints[index].min:=maxint;
end;
end;
procedure dispintarray(var ints:intarray);
var index:integer;
begin
for index:=0 to maxval do
begin
with ints[index] do
write(index:2,':',freq:3,' (',min:5,') (',max:5, ') ');
if( index mod 3=2)then
writeln;
end;
writeln;
end;
begin
randomize;
initintarray(data);
for count:=1 to maxtrials do
begin
rawval:=random(maxint);
result:=rawval mod (maxval+1);
inc(data[result].freq);
if rawval>data[result].max then
data[result].max:=rawval;
```

```
if rawval<data[result].min then
data[result].min:=rawval;
end;
dispintarray(data);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
 0: 46 (  729) (32643)  1: 45 (  514) (32725)  2: 39 (  623) (32294)
 3: 38 (  273) (32214)  4: 40 (  436) (31945)  5: 31 (  275) (32486)
 6: 42 (   60) (32757)  7: 31 ( 1006) (32623)  8: 32 (  899) (32516)
 9: 29 (  792) (32625) 10: 32 (  928) (32680) 11: 35 (  227) (30062)
12: 24 ( 2469) (32034) 13: 41 ( 2848) (32656) 14: 42 ( 1148) (32495)
15: 45 (   15) (31740) 16: 46 ( 1636) (32092) 17: 45 (   44) (32606)
18: 34 ( 1071) (31095) 19: 39 (  289) (32473) 20: 43 (  209) (31988)
21: 35 (  696) (31665) 22: 28 (  859) (32071) 23: 35 (  266) (30776)
24: 36 (  429) (31911) 25: 32 (  322) (30940) 26: 35 (  890) (32534)
```

بنية المعطيات data structure الأساسية في البرنامج السابق هي النسق ذو السبعة و العشرين عنصراً و كل عنصر من عناصر هذا النسق عبارة عن سجل يحتوي على ثلاثة حقول تنتمي إلى نوع المعطيات عدد صحيح integer و الشكل المنطقي التالي يبين لنا كيفية تخزين عناصر هذا النسق:

أسماء حجرات النسق (الدليل)	حجرات النسق
Data[0].freq	
Data[0].min	
Data[0].max	
Data[1].freq	
Data[1].min	
Data[1].max	
....	
....	
....	
Data[maxval].freq	
Data[maxval].min	
Data[maxval].max	

البرنامج السابق يقوم على فكرتين أساسيتين:

1. توليد عدد صحيح عشوائي و من ثم حساب باقي قسمة هذا العدد على العدد 27.
2. تعديل قيمة حجرة النسق و التي دليلها يساوي باقي القسمة السابقة.

و على وجه التحديد يزيد البرنامج السابق عداد تكرار باقي القسمة و يقوم البرنامج بفحص و تعديل إذا احتاج الأمر قيمة الحقلين min و max و اللذين يحتويان على أصغر و أكبر قيمة كان باقي قسمتها على 27 هو دليل الحجرة المحددة فمثلا من خلال الخرج السابق نلاحظ أن 46 قيمة من أصل 1000 قيمة كان باقي قسمتها على 27 هو العدد 0 و أصغر قيمة من هذه القيم هي 729 و أكبر قيمة كانت 32643 .

العمليات على السجلات operations an reords:

يمكننا إلحاق محتويات سجل بشكل كامل إلى متحول من نفس نوع المعطيات (أي نفس السجل) وعملية الإلحاق هذه تحتاج إلى عبارة واحدة فقط لنسخ المساحة من الذاكرة المخصصة لهذا السجل إلى موقع آخر ضمن الذاكرة مخصص للمتحول المراد إلحاق القيمة به و عملية الإلحاق الكلية هذه ناتجة عن حقيقة أن المتحولين ينتميان إلى نوع معطيات واحد أي أن حجم الذاكرة المخصصة لكل منهما متساوٍ بالإضافة إلى عملية الإلحاق الكلية فإن العمليات المطبقة على السجلات تتعلق بأنواع المعطيات التي تتكون منها حقول هذه السجلات لذلك ينطبق على حقل مجموعة العمليات التي يمكن تطبيقها على نوع معطياته.

السجلات المتنوعة variant records:

لنفترض أننا نريد تخزين معلومات حول عدة أشخاص تختلف أعمارهم من الأطفال و حتى البالغين المتزوجين و بالتالي يمكن تخزين الكثير من المعلومات عن هذه المجموعة الواسعة من الأعمار. فعلى سبيل المثال:يمكن تخزين الاسم الأول first name و الكنية last name و العمر age و الجنس sex لكل شخص من هؤلاء الأشخاص و للكن توجد بعض المعلومات التي يتميز بها سن عن آخر فمثلاً : نحن بحاجة إلى معلومات عن الدخل income و عن الوضع العائلي marital status بالنسبة للبالغين adults و بحاجة أيضاً لمعلومات عن حالة المراهقة بالنسبة للمراهقين teenagers أما بالنسبة للأطفال فنحن بحاجة إلى معلومات عن الطفل فيما إذا كان قد التحق بمدرسة أم لا و في حال دخوله المدرسة نحن بحاجة إلى معلومات عن درجات هذا الطفل و إنجازاته المدرسية.

من أجل ذلك يمكننا إنشاء سجلات متنوعة variant records يمكننا من خلالها تخزين مجموعة من المعلومات الخاصة و المتعلقة بنموذج معين مذكور ضمن هذا السجل.

و البرنامج التالي يحتوي على سجل متنوع غايته شرح كيفية إنشاء سجلات المتنوعة:

Code

```
program test;
type
sex=(female,male);
maritalstatus=(single,married,divorecd,widowed);
agegroup=(infant,toddler,child,teenager,adult);
person=record
first,last:string;
age:integer;
gender:sex;
weight,height:real;
case whichgroup:agegroup of
infant:(birthweight,birthheiht:real);
toddler:(inpreschool,indaycare:boolean);
child:(grade:integer);
teenager:(GPA,tIncome:real; tGrade:integer);
adult:(Income:real; status:maritalstatus);
end;
var nemo:person;
begin
writeln('nemo requires ',sizeof(nemo), ' bytes. ');
nemo.income:=39000.50;
writeln(nemo.income:10:2);
nemo.birthweight:=10.50;
nemo.whichgroup:=infant;
nemo.gender:=female;
writeln(sizeof(nemo.whichgroup),' bytes');
case nemo.whichgroup of
infant:writeln('boy infant');
adult:writeln('man');
end;
readln;    end.
```

خرج البرنامج أعلاه هو:

```
nemo requires 542 bytes.
  39000.50
1 bytes
boy infant
```

تبدو التصريحات ضمن البرنامج السابق صعبة إلا أنها في الحقيقية سهلة بسيطة فالسجل person يحتوي على جزأين أحدهما ثابت و الآخر متنوع و الصيغة الكتابية للجزء الثابت ينطبق عليها ما ذكرناه سابقاً بالنسبة لتعريف السجل و في مثالنا السابق هذا الجزء هو الحقول من first و حتى height اما الجزء المتنوع فيبدأ بالعبارة case whichgroubp حتى يستطيع السجل person تخزين خمس مجموعات مختلفة من الحقول حسب قيمة الحقل whichgroup ضمن السجل.
إذا كانت قيمة الحقل nemo.whichgroup هي teenager أي مراهق فإن السجل nemo سوف يخزن ثلاث معلومات فقط و هي tIncome,GPA,tGrade بالإضافة إلى حقول الجزء الثابت أما إذا كانت قيمته هي adult أي بالغ فإن السجل سوف يخزن معلوماتين عن هذا الشخص و هي income و status و بما ان قيمة الحقل whichgroub يمكن تغييرها خلال تنفيذ البرناج فإن البرنامج يجب أن يكون مستعداً لتخزين القيم المناسبة لهذا الحقل أي يجب حجز حجرة من الذاكرة تكفي لتخزين القيم في أسوأ ألاحوال (أكبر مساحة تخزينية) .
فمثلا : السجل السابق preson حجز في الذاكرة مساحة تخزينية قدرها 542 بايت ضمن الذاكرة منها 528 بايت للجزء الثابت و تم توزيعها كما يلي 256 بايت لكل سلسلة رمزية و 2 بايت للحقل age و 6 بايت لكل من height و weight و بايتا واحداً لكل من gender و wghichgroup و أما جزؤه المتنوع فيأخذ 14 بايتاً فقط و هو حجم الذاكرة المطلوب لتخزين أكبر حقل من حقوله المتنوعة و هو teenager حيث يحتاج الحقل infant إلى 12 بايتاً و الحقل toddler إلى 2 بايت و الحقل child يحتاج إلى 2 بايت و الحقل adult يحتاج إلى 7 بايتات ومن خلال ذلك نلاحظ أن 14 bytes التي حجزها السجل person تكفي لتخزين أي معطيات يحتاج إليها السجل لتخزينها و الجدول التالي يوضح حجم الذاكرة المخصصة لكل حقل من حقول السجل person .

الحقل	عنوانه في الذاكرة
First	X
Last	X+256
Age	x+512
Gender	x+514
Weight	x+515
Height	X+521
Whichgroup	x+527

نهاية الجزء الثابت
بداية الجزء المتنوع [حسب قيمة whichgroup]

Birthweight	X+528	[infant]
Inpreschool	X+528	[toddler]
Grade	X+528	[child]
GPA	X+528	[teenager]
Income	X+528	[adult]
Indaycare	X+529	[toddler]
Birthheight	X+534	[infant]
Tincome	X+534	[teenager]
Staus	X+534	[adult]
Tcrade	X+540	[teenager]

السجلات المتنوعة المميزة و غير المميزة tagged and untagged variant:

الحقل whichgroup ضمن التعريفات السابقة يُعرف على أنه حقل تمييز tag field و هذا الحقل يعمل كمرشح filter حيث يمكننا من انتقاء أحد لسجلات المتنوعة إذ يفحص البرنامج قيمة هذا الحقل لتحديد أي سجل من السجلات المتنوعة سوف يقوم بقراءته أو الكتابة فيه أما السجل المتنوع مثل person فيدعى سجلا متنوعاً مميزاً tagged variant لأنه يحتوي على حقل تمييز tag field هو الحقل whichgroup و قد لاحظنا من خلال البرنامج السابق انه لا يمكننا العمل مع القيم المتنوعة من السجل السابق person بدون الحقل whicgroup و في الحقيقة عملية الوصول إلى القسم المتنوع بدون حقل التمييز ليس خطأ تركيبياً و لكنها قد تشكل خطراً في ضياع بعض المعلومات.

ويوجد نوع آخر من انواع السجلات المتنوعة حيث يتجاهل هذا النوع حقل التمييز ولكنه يستعيض عنه بنوع المعطيات المستخدم في تمييز السجلات المتنوعة.

و البرنامج التالي يبين سجل متنوع حذف منه حقل التمييز ولاحظ التغيرات التي حصلت ضمن السطر case في البرنامج التالي:

```
program test;
type
sex=(female,male);
maritalstatus=(single,married,divorecd,widowed);
agegroup=(infant,toddler,child,teenager,adult);
zperson=record
first,last:string;
age:integer;
gender:sex;
weight,height:real;
case agegroup of
infant:(birthweight,birthheiht:real);
toddler:(inpreschool,indaycare:boolean);
child:(grade:integer);
teenager:(GPA,tIncome:real; tGrade:integer);
adult:(Income:real; status:maritalstatus);
end;
var znemo:zperson;
begin
writeln('nemo requires ',sizeof(znemo), ' bytes. ');
znemo.income:=39000.50;
writeln(znemo.income:10:2);
readln;
end.
```

خرج البرنامج أعلاه:

```
nemo requires 541 bytes.
  39000.50
```

لقد تجاهلنا في البرنامج السابق حقل التمييز tag field حيث حذفنا الحقل whicgroup ضمن تعريف السجل فعند تنفيذ البيرنامج السابق فإن متحول السجل znemo يأخذ 541 بايت و هذا الرقم أقل من حجم التخزين الذي أخذ المتحول nemo في البرنامج الذي سبقه ببايت واحد و السبب هو أن السجل znemo لا يتضمن الحقل whichgroup.

استخدام السجلات المتنوعة غير المميزة يتم بواسطة حذف الحقل whichgroup ضمن السجلات المتنوعة المميزة و استخدام السجلات غير المميزة يتطلب كمية تخزين أقل و يجنبنا متطلبات إلى تلك السجلات المتنوعة من حيث فحصه و تعديل قيمة حقل التمييز دائماً و لكننا لا نستطيع تحديد أي الحقول المتنوعة هو المحفز حالياً و هذا طبعاً يقودنا إلى بعض الإرباكات في عملية تخزين القيم ضمن السجلات. و لذلك من الأفضل عند استخدام السجلات المتنوعة اختيار السجلات المتنوعة المميزة.

و يجدر الإشارة هنا إلى أن عبارة case (و التي تأتي دليلاً على أن هذا السجل هو سجل متنوع و هي أيضاً تعرف حقل التمييز) يجب ان تأتي كآخر تعريف ضمن حقول السجل المتنوع و عملية تداخل السجلات المتنوعة ضمن بعضها أي تعريف حقل تمييز ضمن حقل تمييز آخر أمر غير مسموح به أي لا يمكن وضع أكثر من حقل تمييز واحد ضمن السجل المتنوع الواحد.

السجلات الثابتة record constant :

يمكننا تعريف سجلات ثابتة في لغة التربو باسكال أي يمكننا تعريف سجل و تحديد اسم لهذا السجل ليكون هذا الاسم مرتبطاً تماما بمجموعة من القيم المخزنة في هذا السجل و هذه القيم لن تتغير طوال فترة تنفيذ البرنامج و البرنامج التالي يبين لنا كيفية تعريف السجلات الثابتة:

Code

```
program test;
type dailyweather=record
    hitemp,lotemp,wind:real;
    end;
daysdata=array[1..2] of real;
daysweather=record
        dayshi,dayslo,dayswind:daysdata;
        end;
const extremeweather:dailyweather=(hitemp:115.6;lotemp:-89.9;wind:183.2);
extremedays:daysweather=(dayshi:(101.6,115.6);dayslo:(-89.9,-88.7);
                    dayswind:(183.2,181.8));
begin
with extremeweather do
begin
writeln('hitemp  = ',hitemp:12:2);
writeln('lotemp  = ',lotemp:12:2);
writeln('wind    = ',wind  :12:2);
end;
writeln;
with extremedays do
begin
writeln('<data>       <midnight>    <noon>');
writeln('dayshi    = ',dayshi[1]:12:2,dayshi[2]:12:2);
writeln('dayslo    = ',dayslo[1]:12:2,dayslo[2]:12:2);
writeln('dayswind   = ',dayswind[1]:12:2,dayswind[2]:12:2);
end;
readln;
end.
```

خرج البرنامج السابق هو

```
hitemp  =       115.60
lotemp  =       -89.90
wind    =       183.20

<data>          <midnight>      <noon>
dayshi      =       101.60      115.60
dayslo      =       -89.90      -88.70
dayswind    =       183.20      181.80
```

يمكننا تعريف سجل ثابت ضمن القسم const بواسطة تحديد مميز هو اسم لهذا السجل و من ثم يُتبع هذا المميز بنوع المعطيات و بمجموعة من القيم و الصيغة الكتابية لتعريف سجل ثابت هي:

<قيم > = <نوع معطيات سجل > :<مميز>

و تحدد قيم السجل الثابت ضمن أقواس كما هو الحال ضمن الأنساق الثابتة إذ يذكر اسم الحقل متبوعاً بنقطتين ثم بقيمة هذا الحقل و الصيغة الكتابية لتعريف القيم هي:

قيمة :اسم الحقل

ينبغي علينا عند وضع قيم السجل الثابت أن نضع هذه القيم وفق الترتيب الصحيح لحقول السجل كما وردت ضمن تعريف السجل الأصلي ، فاصلين بين حقول السجل بفاصلة منقوطة.

و لذلك حددنا ضمن السجل الثابت extremeweather ثلاثة حقول هي أسماء حقول السجل الأصلي و أتبعت هذه لاالأسماء بنقطتين ومن ثم بالقيمة المراد تخزينها في هذا الحقل أما السجل الثابت extremdays و الذي حقوله عبارة عن أنساق كانت عملية تحديد قيم السجل تخضع لاتحاد طريقتي تهيئة الأنساق الثابتة و السجلات الثابتة إذ وضع قيم الحقل ضمن قوسين و ألحقت باسم الحقل.

نوع المعطيات كائن: the object type:

تمكننا لغة التربو باسكال من العمل وفق تقانة الرمجة غرضية التوجه (القوية) oop أي البرمجة كائنية التوجه object oriented programming و هذه التقانة سوف أشرحها بشكل تفصيلي في الجزء الثاني. حيث يتألف الكائن object بمجموعة من الميزات features أو الخواص property مهمتها وصف هذا الكائن فمثلاً يمكن ان نصف الكرة على أنها كائن يملك حجماً و قطراً و هذه الكرة قد تكون مفرغة أو مصمتة. بعد تعريف الكائن الذي نريد يمكن تعريف كائن آخر يملك نفس صفات الكائن الأول مع بعض الصفات أو الخواص الإضافية فمثلاً : يمكن وصف كرة القدم بنفس صفات الكرة التي وصفناها سبقاً مع بعض المواصفات الإضافية منها اسم هذه الكرة و الرسومات الموجودة على سطحها و ما إلى ذلك.
تمثل الكائنات في لغة التربو باسكال بطريقة تشبه تمثيل السجلات لذلك يجوز أن نطلق على الكائنات اصطلاحاً بالسجلات الكائنية .
يمكننا تمثيل الكائن الكرة وفق الخاصتين التاليتين:

```
Type sphere=object
              radius:real;
              solid:Boolean;
             end;
```

و التصريح التالي يبين لنا كيفية التصريح عن نموذج للكائن sphere :

```
Var Euclideansphere : sphere;
```

و عندما نعرف كائناً ما يمكننا تعريف أنواع جديدة من هذا الكائن و هذه الأنواع ترث inherit خواص هذا الكائن مع أنها تملك خواصها الخاصة بها. و التعريفات التالية توضح لنا الكائنات المشتقة من الكائن الأصلي sphere :

```
Type sphere=object
              radius:real;
              solid:Boolean;
             end;
design=(solidcolor,patterns,pictures);
beachball=object (sphere)
           surface:design;
           end;
sportball=object (sphere)
          bouncy: Boolean;
         end;
var mybeachball:beachball;
      squashball, basketball:sportball;
```

الكائنات التي ترث ميزاتها من كائنات أخرى تدعى بالكائنات الخلف descendant objects و في مثالنا كانت الكائنات beachball و sportball هي كائنات اخلاف و هذه الكائنات تملك أباً يدعى بالكائن السلف ancestor object و في مثالنا هو الكائن sphere و الكائنات الأخلاف يمكن أن تملك كائنات متفرعة عنها أيضاً (هكذا الحياة).
من أجل تعريف الكائنات الخلاف نستخدم التعريف object نفسه مع إضافة اسم الكائن السلف بين قوسين بعد المميز المحجوز object و من خلال هذا التعريف يسمح للكائن الخلف بأن يرث كل صفات الكائن السلف أي حقول الكائن السلف تستخدم ضمن الكائن الخلف كما لو أنها عرفت ضمنه.
والصيغة الكتابية لتعريف الكائنات و الكائنات الأخلاف هي:

```
< اسم الكائن > = object
        < محتويات الكائن >
        End;
< اسم الكائن الخلف > = object (اسم الكائن السلف)
            < محتويات الكائن >
            End;
```

و عملية الوصول إلى حقول الكائنات تشبه إلى حد بعيد الوصول إلى حقول السجلات و العبارات التالية مثال على عملية الوصول إلى بعض حقول الكائنات السابقة:

```
Squashball.solid:=false;
Squashball.bouncy:=true;
Squashball.radius:=12;
Squashball.surface:=pictures;
```

و نكمل شرح هذه التقانة البرمجية إلى الجزء الثاني من الكتاب.

نوع المعطيات مجموعة the set type :

قد تحتاج في بعض الأحيان إلى تجميع عدة قيم لاستخدامها أثناء عمليات المقارنة . لنفرض أننا نريد فحص عدد الأحرف التي يكتب جزء منها تحت السطر عند طباعة سطر ما أي أننا نريد ان نفحص عدد مرات تكرار الأحرف y,g,p,j,q او لنفرض أننا نريد تتبع اختصاصات بعض الطلبة بافضافة إلى تحديد عددهم في كل اختصاص عندئذ علينا تحديد عدد الطلاب في كلا الاختصاصين العلوم science أو الدراسات الثقافية الإنسانية humanity ومن اجل اختصاص العلوم علينا تحديد اختصاص الطلبة هل هو رياضيات math أو فيزياء physics أو كيمياء chemistry أو علم الأحياء biology أو الكيمياء الحيوية biochemistry أو الفلك astronomy او علم الأرض earth science وضمن هذه الفروع نلاحظ اننا لا نريد التعامل لا نحتاج إلى التعامل مع قيم بل مع مجموعات من القيم و اعتمادا على المثال السابق يمكن أن تحتوي المجموعة الواحدة على قيم من نوع معطيات معين.

و تعريف المجموعة رياضياً عبارة عن عدد من القيم او العناصر غير المرتبة و غير المكررة و المتجمعة فيما بينها و يمكن تمثيل المجموعة رياضياً بواسطة سرد عناصرها جميعاً و بدون تكرار .

لنفترض أن لدينا السلسلة الرمزية sweeteners و اننا نريد تمثيل رموز هذه السلسلة ضمن مجموعة فإن هذه المجموعة سوف تحتوي على الرموز التالية : r,s,t,n,e,w,s مع ملاحظة ان الحرفين s,e قد وضعا ضمن المجموعة مرة واحدة فقط في حين أنهما ظهرا ضمن السلسلة أكثر من مرة.

وظيفة نوع المعطيات مجموعة set في لغة باسكال هي تمثيل المجموعات و البرنامج التالي يبين لنا كيفية تعريف و تهيئة المجموعات :

Code

```
program test;
type charset=set of char;
major=(anthropology,art,astronomy, biochemistry, biology,chemistry,complit,earthsci,
    economics, english, literature, math,physics,politics,psychology);
majorgroup=set of major;
var allmajors,science:majorgroup;
   wdset:charset;
begin
allmajors:=[anthropology..psychology];
science:=[physics,astronomy..chemistry,earthsci];
wdset:=['b','d','f','h','k','l','t','u'..'z'];
readln;
end.
```

و المجموعات كنوع من أنواع المعطيات المركبة لابد أولا من تعريفها ضمن قسم التعريف و الصيغة الكتابية لنوع المعطيات مجموعة :

```
<مميز> = set of < نوع معطيات أساسي ترتيبي >
```

و يمكن تعريف بعد اللكمة المحجوزة var :

```
Var < مميز>: set of < نوع معطيات ترتيبي >
```

و في البرنامج السابق عرفنا النوع charset على أنه مجموعة من العناصر التي تنتمي إلى النوع char .

أما النوع majorgroup فهو يعرف نوع معطيات مجموعة نوع معطياتها الأساسي base type من النوع التعدادي major .

و عملية إلحاق القيم بمتحول المجموعة موضح في البرنامج السابق حيث استخدمنا معامل الإلحاق العادي بالإضافة إلى باني المجموعة set constructor أي القوسان المربعان [ ] اللذان يحددان بينهما عناصر المجموعة فعلى سبيل المثال : التركيب التالي ينشئ مجموعة تحتوي على 15 قيمة من نوع المعطيات major :

allmajors:=[anthropology..psychology];

لتحديد عناصر المجموعة علينا ببساطة سرد عناصر هذه المجموعة ضمن قوسين مربعين مع إمكانية استخدام النقطتين المتتاليتين لتمثيل مجال من القيم المتتالية .

القيود المفروضة على المجموعات set restriction :

يجب أن يكون نوع المعطيات الأساسي الخاص بالمجموعة نوعاً ترتيبياً يملك 256 قيمة كحد أعظمي أي أن قيم النوع الترتيبي لنوع المعطيات الأساسي للمجموعة لا يمكن أن يكون أقل من 0 أو أكثر من 255 و هذا يعني أننا لا نستطيع كتابة التعريف التالي:

Set of integer;

لأن هذا النوع يحدد قيماً ممكنة أكثر من 256 و كذلك الأمر بالنسبة للتعريف التالي:

Set of shortint;

بسبب كون الأعداد التي ينتمي إليها هذا النوع هو من 128- إلى 127 أي يوجد بعض القيم التي هي أقل من 0 و هذا عير مسموح به و كذلك الأمر بالنسبة للتريفين التاليين:

Set of word;
Set of longint;

اما إذا انشانا نوع معطيات تعدادي فيجب أن لا يتجاوز عدد قيم هذا النوع أكثر من 256 قيمة إذا أردنا استخدام هذا النوع كنوع معطيات أساسي للمجموعة.

يمكننا إنشاء نوع معطيات مجموعة من نوع معطيات مجال الجزئي فعلى سبيل المثال: التعريف التالي يعرف مجموعة عناصرها يملك أن تكون أي قيمة بين 20 و 199 :

Code
```
program test;
type middlenrs=20..199;
   middleset=set of middlenrs;
var midvals:middleset;
begin
midvals:=[20,199,36];
readln;
end.
```

إذاً نوع المعطيات الأساسي base type لنوع المعطيات مجموعة يجب أن يكون نوعاً ترتيبياً و يحقق الشرطين التاليين:

1. أن لا تتجاوز قيم نوع المعطيات الأساسي أكثر من 256 قيمة.
2. أن يكون الموضع الترتيبي لهذه القيم بين المجال من 0 و حتى 255 .

تمثيل المجموعات representation of sets :

يمكننا تمثيل المجموعات في لغة التربو باسكال بنسق من الخانات الثنائية bits تقابل كل خانة موضع قيمة في المجموعة فإذا كانت قيمة هذه الخانة هي 1 أي أن العنصر موجود ضمن المجموعة أما إذا كانت قيمة هذه الخانة 0 فإن العنصر المقابل لا ينتمي إلى المجموعة و تمثيل المجموعات هذا يفسر لنا عدم الاهتمام بترتيب عناصر المجموعة و يفسر أيضاً لماذا كان تكرار العنصر أمراً لايؤثر على المجموعة.

لنفترض أن لدينا البرنامج التالي الذي يعرف النوع threedigit على أنه مجموعة من نوع مجال جزئي fivedigit كما يلي:

Code
```
program test;
type fivedigit=1..5;
   threedigit=set of fivedigit;
var three:threedigit;
begin
three:=[1,3,5];
```

```
end.
```

نلاحظ من خلال البرنامج أعلاه أننا أعطينا المجموعة three ثلاث قيم هي 1 و 3 و 5 و عندئذ يمكن تمثيل هذه المجموعة بالنسق التالي مع العلم بأن الحجرة التي فيها 1 تدل على أن دليل هذه الحجرة ينتمي إلى المجموعة three أما الحجرة التي فيها 0 فتدل على أن دليلها لا ينتمي إلى المجموعة three :

	three
1	1
2	0
3	1
4	0
5	1

مجموعة خاصة A Special set :
يوجد مجموعة خاصة جداً و هي المجموعة التي لا تملك أي عنصر و تدعى بالمجموعة الفارغة empty set أو المجموعة الخالية null set .
و هذه المجموعة تشبه الرقم 0 في الرياضيات حيث إنه ضروري و لكن ليس له تأثير على كثير من العمليات الرياضية و تحدد المجموعة الخالية بواسطة قوسين مربعين فارغين أي [ ] .

```
Three:=[];
```

معاملات المجموعات set operators:

يمكن إضافة عناصر إلى المجموعة أو اختيار عنصر ما أو حذف عنصر ما و يمكن مقارنة مجموعنين فيماإذا كانتا تحتويان نفي العناصر أم لا و يمكننا فحص فيما قيمة معينة تنتمي إلى نفس المجموعة أم لا.

اختيار وجود عنصر في مجموعة checking wether an element is in a set :

تزودنا لغة باسكال بالمعامل in الذي ي}كل إليه فحص قيمة ما هل هي محتواة ضمن المجموعة أم لا و هذا المعامل الثنائي يأخذ binary operator يأخذ حدين مختلفين الأول هو عنصر من روع المعطيات الأساسي للمجموعة و الحد الثاني هو المجموعة ذاتها و يعيد هذا المعامل القيمة قيمة بوليانية هي true إذا كان العنصر ينتمي إلى المجموعة و false إذا لم ينتمي العنصر إلى المجموعة.
و البرنامج التالي يوضح لنا استخدام المعامل in :

Code

```
program test;
type charset=set of char;
var wdset:charset;
procedure dispcharset(theset:charset;startch,endch:char);
var indexch:char; count:integer;
begin
count:=1;
for indexch:=startch to endch do
if indexch in theset then
begin
write(indexch:3);
if count mod 20=0 then writeln;
inc(count);
end;
end;
begin
wdset:=['A'..'z'];
dispcharset(wdset,'A','z');
readln;
end.
```

خرج البرنامج السابق هو:

```
 A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T
 U  V  W  X  Y  Z  [  \  ]  ^  _  `  a  b  c  d  e  f  g  h
 i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
```

معامل اجتماع المجموعات the set union operator :

يزودنا معامل اجتماع المجموعات union operator و الذي يُدعى أيضاً بمعامل الجمع addition operator بإمكانية إضافة مجموعة إلى أخرى و البرنامج التالي يبين لنا مثالاً عن هذا المعامل:

Code

```
program test;
type charset=set of char;
var wdset1,wdset2 :charset;
procedure dispcharset(theset:charset;startch,endch:char);
var indexch:char; count:integer;
begin
count:=1;
for indexch:=startch to endch do
if indexch in theset then
begin
write(indexch:3);
if count mod 20=0 then writeln;
inc(count);
end;
end;
begin
wdset1:=['b','d','f','h','k','l','t'];
wdset2:=wdset1+['g','j','p','q','y'];
writeln('wdset1: ');
dispcharset(wdset1,'a','z');
writeln;
writeln('wdset2: ');
dispcharset(wdset2,'a','z');
readln;
end.
```

خرج البرنامج السابق هو:

```
wdset1:
  b  d  f  h  k  l  t
wdset2:
  b  d  f  g  h  j  k  l  p  q  t  y
```

يبني البرنامج السابق المجموعة wdset2 عبر دمج محتويات مجموعتين إذ يأخذ معامل اجتماع المجموعات مجموعتين كحدود له و يعيد مجموعة ثالثة تدعى ناتج اجتماع مجموعتين و هي تحتوي على العناصر الموجودة في كلتا المجموعتين و بدون تكرار للعناصر طبعاً.

و لابد من التذكر دائماً أن تكرار العناصر غير مسموح به ضمن المجموعات و هذا يعني أن العنصر الموجود في كلتا المجموعتين تشكلان حدي معامل الدمج سوف يوضع مرة واحدة في المجموعة الناتجة عن هاتين المجموعتين. فمثلا عبارة الإلحاق التالية ستلحق نفس العناصر بالمجموعة wdset2 التي تلحقها عبارة الإلحاق الموجودة ضمن البرنامج السابق :

wdset2:=wdset1+['g','j','p','q','y']+['g','y','y'];

توجد في عبارة الإلحاق السابقة مجموعة تكرر فيها ذكر العنصر y و هذا التكرار لن يسبب خطأ في عملية الترجمة أو تغييراً في المجموعة الناتجة عن دمج هذه المجموعات ولذلك نقول أن تكرار عنصر ما في مجموعة ليس له أي تأثير و اي معنى و

المجموعة الناتجة عن دمج + المجموعات سيحتوي على الأقل عددا من العناصر يساوي عدد العناصر الموجودة في أكبر مجموعة من هذه المجموعات.
ملاحظة: ناتج دمج (اجتماع) المجموعة الخالية مع مجموعة ما هو المجموعة نفسها.

معامل تقاطع المجموعات the set intersection operator :
ينشئ مجموعة عناصرها منتقاة selected من كلتا المجموعتين الطبق عليهما معامل التقاطع و معامل التقاطع هذا ثنائي binary operator يأخذ حدين عبارة عن مجموعتين و يعيد مجموعة ناتجة عن تقاطع هاتين المجموعتين تحتوي فقط على العناصر المحتواة في كلتا المجموعتين و عدد عناصر هذه المجموعة طبعاً سيكون على الأكثر يساوي عدد أصغر مجموعة من المجموعتين المطبق عليهما معامل التقاطع.

Code

```
program test;
const smaller=3;
        larger=4;
type byteset=set of byte;
var bysmaller,bylarger,result:byteset;
procedure dispbyteset(theset:byteset;startbyte,endbyte:byte);
var indexbyte:byte; count:integer;
begin
count:=1;
for indexbyte:=startbyte to endbyte do
if indexbyte in theset then
begin
write(indexbyte:3);
if count mod 20=0 then writeln;
inc(count);
end;
end;
procedure showsetdata(theset:byteset; message:string);
begin
write(message, ':');
dispbyteset(theset,0,255);
writeln;
end;
procedure buildset(var theset:byteset;increment:integer);
var count, val:integer;
begin
val:=0;
for count:=1 to 20 do
begin
inc(val,increment);
theset:=theset+[val];
end;
end;
begin
bysmaller:=[];
bylarger:=[];
buildset(bysmaller,smaller);
buildset(bylarger,larger);
showsetdata(bysmaller,'bysmaller');
showsetdata(bylarger,' bylarger');
result:=bysmaller * bylarger;
```

```
showsetdata(result,'bysmaller * bylarger');
readln;
end.
```

خرج البرنامج السابق هو:

```
bysmaller:  3  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60
 bylarger:  4  8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80
bysmaller * bylarger: 12 24 36 48 60
```

يستخدم البرنامج السابق معامل الاجتماع لبناء مجموعتين من القيم بين 0 و 255 تحتوي المجموعة الأولى على مضاعفات العدد 3 و الثانية على مضاعفات العدد 4 و تتم هذه الأعمال ضمن الإجراء buildset الذي يزودنا أيضاً عن طريقة تمرير المجموعات مرجعياً و ذلك بذكر اسم المجموعة المراد تمريرها مسبوقة بالمميز var و عندئذ سوف يربط البرنامج الحجرات المخصصة لتخزين هذه المجموعة بالمتحول الوسيطي argument المخصص لها و بالتالي أي تغييرات تجري في قيمة المتحول الوسيط لن تتلاشى بإنهاء تنفيذ الإجراء بل ستبقى و تعاد إلى البرنامج الرئيسي ليتعامل مع المجموعة وفق قيمها الجديدة و يظهر الإجراء showsetdata محتويات المجموعة الممررة إليه ابتداء من اسم هذه المجموعة و من ثم يستدعي إجراء آخر اسمه dispbyteset .

و في البرنامج الرئيسي المجموعة result تحتوي على تقاطع المجموعتين bysmaller , bylarger أي تحتوي على مضاعفات العدد 12.

نلاحظ من خلال الخرج السابق أن المجموعة الناتجة عن تقاطع المجموعتين السابقتين كانت تملك أقل عدداً من العناصر مقارنة مع المجموعتين اللتين طبق عليهما معامل التقاطع و في حال عدم وجود عدم تطابق no overlap بين عناصر المجموعتين (هذا يعطينا بعد تطبيق معامل التقاطع مجموعة خالية) عندئذ يطلق على هاتين المجموعتين بالمجموعتين الخاليتين Disjoint set و البرنامج التالي يعطينا مثالاً عن مجموعتين منفصلتين هما : doubeldigit , singledigit :

Code

```
program test;
type byteset=set of byte;
var singledigit,doubledigit,overlap:byteset;
count, nrintersects:integer;
begin
nrintersects:=0;
singledigit:=[0..9];
doubledigit:=[10..99];
overlap:=singledigit*doubledigit;
writeln('overlap: ');
for count:=0 to 255 do
begin
if count in overlap then
begin
write(count:4);
inc(nrintersects);
end;
end;
writeln;
writeln(nrintersects,' common values');
readln;
end.
```

خرج البرنامج السابق هو:

المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com
Syrian Arab Republic
دمشق-معضمية الشام

```
overlap:
0 common values
```

نلاحظ من خلال خرج البرنامج السابق أن المجموعة overlap لا تحتوي على عناصر لذلك لن يظهر أي شيء ما عدا قيمة المتحول nrintersects و بالتالي فإن تقاطع مجموعتين منفصلتين بالتعريف هو مجموعة خالية و تقاطع مجموعة خالية مع مجموعة خالية هو مجموعة خالية أيضاً. و المجموعة الخالية تكافئ الصفر في عملية الضرب و معامل التقاطع يشبه معامل الضرب من حيث أسبقيته precedence فهو يملك الأسبقية ذاتها التي يملكها معامل الضرب و المعاملات المنطقية.

معامل فرق المجموعات the set difference operator :

لنفترض أننا نريد معرفة عدد المواد التي أخذها طالب ما من جميع المواد المقررة عليه عندئذ علينا فحص المواد المقررة مادة مادة لمعرفة أيها اخذها الطالب و أيها لم يأخذها و يُمكننا معامل الفرق من معالجة هذه المسألة إذ يأخذ هذا المعامل الثنائي حدين عبارة عن مجموعتين و يعيد مجموعة ثالثة . ففي مثالنا إذا فرضنا مجموعة المواد التي أخذها الطالب هي: studentscourses و المواد المفروضة عليه هي requiredcourded عندئذ يمكننا كتابة التعبير:

Requiredcourses – studentscourses

فإذا كان ناتج هذا التعبير أي قيمة عدا المجموعة الخالية عندئذ يوجد هناك بعض المواد لم يأخذها الطالب بعد و البرنامج التالي معامل الفرق:

Code

```
program test;
const smaller=6;
    larger=8;
type byteset=set of byte;
var bysmaller,bylarger,result:byteset;
count:integer;
procedure dispbyteset(theset:byteset;startbyte,endbyte:byte);
var indexbyte:byte; count:integer;
begin
count:=1;
for indexbyte:=startbyte to endbyte do
if indexbyte in theset then
begin
write(indexbyte:4);
if count mod 15=0 then writeln;
inc(count);
end;
end;
procedure showsetdata(theset:byteset; message:string);
begin
write(message, ':');
dispbyteset(theset,0,255);
writeln;
end;
procedure buildset(var theset:byteset;increment:integer);
var count, val:integer;
begin
val:=0;
for count:=1 to 20 do
begin
inc(val,increment);
```

```
theset:=theset+[val];
end;
end;
begin
bysmaller:=[];
bylarger:=[];
buildset(bysmaller,smaller);
buildset(bylarger,larger);
showsetdata(bysmaller,'bysmaller');
showsetdata(bylarger,' bylarger');
result:=bysmaller - bylarger;
showsetdata(result,'bysmaller - bylarger');
readln;
end.
```

خرج البرنامج السابق هو:

```
bysmaller:   6  12  18  24  30  36  42  48  54  60  66  72  78  84  90
  96 102 108 114 120
 bylarger:   8  16  24  32  40  48  56  64  72  80  88  96 104 112 120
 128 136 144 152 160
bysmaller - bylarger:   6  12  18  30  36  42  54  60  66  78  84  90 102 108 114
```

يبني البرنامج السابق مجموعتين تحتوي الأولى على مضاعفات العدد 6 و الثانية تحتوي على مضاعفات العدد 8 و بعد إظهار محتويات هاتين المجموعتين يحذف البرنامج مضاعفات العدد 8 في المجموعة التي تحتوي على مضاعفات العدد 6 أي مضاعفات العدد 6 و 8 المشتركة سوف تحذف من المجموعة bysmaller.

إذا كانت المجموعتان المطبق عليهما معامل الفرق منفصلتين disjoint فإن المجموعة التي يجب حذف بعض عناصرها المشتركة مع المجموعة الأخرى لن تتغير و البرنامج التالي يوضح لنا فكرة حذف العناصر من المجموعات إذ إن الجزء الأول في هذا البرنامج يحذف رمزين من المجموعة أما الجزء الثاني فيسمح لنا بحذف أي رمز نختاره.

Code

```
program test;
type charset=set of char;
var wdset:charset;
   thech:char;
procedure dispcharset(theset:charset;startch,endch:char);
var indexch:char; count:integer;
begin
count:=1;
for indexch:=startch to endch do
if indexch in theset then
begin
write(indexch:3);
if count mod 20=0 then writeln;
inc(count);
end;
writeln;
end;
procedure getchar(message:string; var value:char);
begin
write(message,' ');
readln(value);
end;
```

```
begin
wdset:=['a','c'.. 'x','z'];
dispcharset(wdset,'a','z');
wdset:=wdset-['a'];
writeln('After removing ''a''');
dispcharset(wdset,'a','z');
wdset:=wdset-['b'];
writeln('After removing ''b''');
dispcharset(wdset,'a','z');
getchar('remove what char? (! to stop)',thech);
while(thech<>'!')do
begin
wdset:=wdset-[thech];
writeln('After removing ''',thech,'''');
dispcharset(wdset,'a','z');
getchar('remove what char? (! to stop)',thech);
end;
end.
```

خرج البرنامج السابق هو:

```
 a c d e f g h i j k l m n o p q r s t u v w x z
After removing 'a'
 c d e f g h i j k l m n o p q r s t u v w x z
After removing 'b'
 c d e f g h i j k l m n o p q r s t u v w x z
remove what char? (! to stop) x
After removing 'x'
 c d e f g h i j k l m n o p q r s t u v w z
remove what char? (! to stop) !
```

بعد أن يبني البرنامج المجموعة wdset يحذف رمزين هما a و b و نلاحظ من خلال الخرج السابق ان المجموعة بعد حذف الحرف b لم تتغير عن المجموعة نفسها بعد أن حذفنا منها الحرف a و ذلك لأن الحرف b لم يكن موجوداً أصلاً ضمن هذه المجموعة. معامل الفرق يملك نفس أسبقية معامل الاجتماع و بالتالي نفس أسبقية معامل الجمع.

و نلاحظ من خلال الخرج السابق كيف استطعنا إظهار الفاصلة العلوية ' في خرج البرنامج و ذلك ضمن استدعائي الإجراء writeln اللذان يحتويان على السلسلة الرمزية After removing حيث استخدمنا زوجين من الفواصل العلوية لإظهار هذه الفواصل على الشاشة و لنأخذ على سبيل المثال :

writeln('After removing ''b''');

عند وضع فاصلتين علويتين متتاليتين فهذا يعني أننا نريد إظهار زوج الفواصل الداخلي على الشاشة و الفاصلة الثالثة بعد الحرف b تعني إنهاء للسلسلة الرمزية المررة كمتحول وسيطي للإجراء writeln.

استخدام الإجراءات لإضافة أو حذف عناصر من المجموعات

using procedures to add and remove set elements:

تزودنا لغة التربو باسكال و في الإصدار السابع تحديداً بإجراءين جديدين للتعامل مع المجموعات و هذان الإجراءان هما include و exclude و يأخذ كل إجراء من هذين الإجراءين متحولين وسيطين حيث المتحول الأول عبارة عن مجموعة يمكن ان يكون نوع معطياتها الأساسي أي نوع من أنواع المعطيات المسموحة و التي ذكرناها آنفاً و المتحول الثاني متحول ينتمي لنفس نوع المعطيات الأساسي الخاص بالمجموعة.

فالإجراء include يضيف عنصراً واحداً وهو قيمة المتحول الوسيطي الثاني الممرر إليه (أي يضيف هذا العنصر إلى المجموعة الممرة إليه).

أما الإجراء exclude فيحذف عنصراً واحداً من المجموعة و هذا العنصر هو قيمة المتحول الممرر إليه.

و البرنامج التالي يبين لنا كيفية عمل هذان التابعان:

Code

```
program test;
var s1:set of 1..30;
i:integer;
procedure printset;
begin
for i:=1 to 30 do
if(i in s1 )then
write(i,', ');
writeln;
end;
begin
s1:=[10,5,20];
include(s1,30);
include(s1,21);
include(s1,5);
printset;
exclude(s1,30);
exclude(s1,5);
writeln;
printset;
readln;
end.
```

خرج البرنامج السابق هو:

```
5, 10, 20, 21, 30,

10, 20, 21,
```

مقارنة المجموعات comparing sets :

تملك لغة التربو باسكال أربعة معاملات تستخدم لمقارنة المجموعات و عملية المقارنة بين المجموعات تعتمد على العناصر في هذه المجموعات بالإضافة إلى عدد عناصر هذه المجموعة فعلى سبيل المثال يمكننا مقارنة مجموعتين لتحديد فيما إذا كانتا تملكان نفس العناصر حيث يطلق على هاتين المجموعتين بالمجموعتين المتساويتين equal sets و يمكننا من خلال مقارنة المجموعات تحديد في ما(= فيما)إذا كانت المجموعة A مثلاً تحتوي على كل العناصر المحتواة في المجموعة B عندئذ يمكننا القول بأن المجموعة A هي المجموعة الأساسية superset للمجموعة B في حين يقال عن المجموعة B أنها مجموعة جزئية subset من المجموعة A . ومعاملات المقارنة هذه و ككل معاملات المقارنة سوف تعيد قيماً بوليانية وحدود هذه المعاملات طبعاً مجموعات لذلك نقول أن معاملات المقارنة تأخذ حدين عبارة عن مجموعتين كمتحولات وسيطية و تعيد قيمة بوليانية.

معامل تساوي و معامل عدم تساوي المجموعات the set equality and inequality operators :

تزودنا لغة باسكال بمعامل التساوي (=) equality operator الذي يمكن من فحص تساوي مجموعتين تماماً حيث يعيد true في حال تساوي المجموعتين أي احتواء المجموعتين على نفس العناصر تماماً و إلا يعيد القيمة false .

ليكن لدينا المجوعتين التاليتين:

```
Set1:=['a','c'..'m','p'];
Set2:=['a'..'m','p'..'s'];
```

معامل المساواة في عبارة if التالية سوف يعيد القيمة false :

```
If set1=set2 then
```

و يوجد في لغة باسكال أيضاً معامل عدم المساواة <> inequality operator الذي يفحص اختلاف المجموعات (عدم تساويها)

معامل المجموعة الجزئية the subset operator :

نلاحظ من خلال المثال السابق أن عناصر المجموعة set1 موجودة أيضاً ضمن المجموعة set2 و يمكننا تحديد هذه العلاقة بين هاتين المجموعتين بواسطة معامل المجموعة الجزئية subset operator و عبارة if التالية تظهر لنا كيفية تحديد فيما إذا كانت المجموعة set1 هي مجموعة جزئية من set2 أم لا:

If set1<=set2 then

و قيمة التعبير البولياني في هذه الحالة هو true .

معامل المجموعات الأساسية the superset operator :

عندما نقول أن المجموعة set 1 هي مجموعة جزئية subset من المجموعة set2 فإن قولنا هذا مكافئ تماماً لقولنا أن المجموعة set2 هي مجموعة اساسية superset للمجموعة set1. و في لغة باسكال معامل المجموعة الأساسية يزودنا بإمكانية فحص هذه العلاقة بين المجموعات و العبارة التالية مكافئة تماما لسابقتها التي تستخدم معامل المجموعة الجزئية .

If set2>=set1 then

و اعتماداً على تعريف المجموعة الجزئية فإن المجموعتين المتساويتين هما مجموعتان جزئيتان من بعضهما أي أن الأولى جزئية من الثانية و الثانية من الأولى.

و البرنامج التالي يبين لنا أن قيمة التعابير البوليانية الموجودة في عبارة if سوف تكون true إذا كانت المجموعتان set1 و set2 متساويتين:

Code

```
program test;
type charset=set of char;
var set1,set2,alphabet:charset;
procedure printset(theset:charset);
var c:char;
begin
for c:='a' to 'z' do
if c in theset then
write(c,', ');
writeln;
end;
begin
alphabet:=['a'..'z'];
set1:=['a'..'l','o'..'z'];
set2:=alphabet-(alphabet-set1);
printset(alphabet);
printset(set1);
printset(set2);
if set1=set2 then
writeln('set1=set2 ',set1=set2);
if(set1<=set2)then
writeln('set1<=set2 (set1 is a subset of set1 ',set1<=set2);
if(set1>=set2)then
writeln('set1>=set2 (set1 is a superset of set1 ',set1>=set2);
readln;
end.
```

خرج البرنامج السابق هو:

```
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,
a, b, c, d, e, f, g, h, i, j, k, l, o, p, q, r, s, t, u, v, w, x, y, z,
a, b, c, d, e, f, g, h, i, j, k, l, o, p, q, r, s, t, u, v, w, x, y, z,
set1=set2 TRUE
set1<=set2 (set1 is a subset of set1 TRUE
set1>=set2 (set1 is a superset of set1 TRUE
```

يستخدم البرنامج السابق ثلاثة معاملات مقارنة من اجل مقارنة المجموعتين set1 و set2 و عبارتي الإلحاق في بداية البرنامج تلحقان نفس العناصر بالمجموعتين.
و الجدول التالي يبين لنا المعاملات الطبقة على نوع المعطيات مجموعة:

المعامل operator	الحد اليساري left operator	الحد اليميني Right operator	الناتج redult
in	قيمة	مجموعة	Boolean
الاجتماع +	مجموعة	مجموعة	مجموعة
التقاطع *	مجموعة	مجموعة	مجموعة
الفرق -	مجموعة	مجموعة	مجموعة
التساوي =	مجموعة	مجموعة	Boolean
عدم التساوي <>	مجموعة	مجموعة	Boolean
المجموعة الجزئية <=	مجموعة	مجموعة	Boolean
المجموعة الأساسية >=	مجموعة	مجموعة	Boolean

المجموعات الثابتة و المجموعات عديمة الأسماء set constants and anonymous sets :

يمكننا تعريف مجموعات ثابتة في لغة التربو باسكال و البرنامج التالي يوضح لنا كيفية تعريف و تهيئة المجموعات الثابتة بعد تهيئتها يمكن الإشارة إليها بذكر اسمها و لكن لا نستطيع تطبيق أي عملية عليها لتغيير محتوياتها طبعاً.

Code

```
program test;
type charset=set of char;
const smallletters:charset=['a','c','e','i','m'..'o','r','s','u'..'x','z'];
singledigitset:set of byte=[0..9];
doubledigitset:set of 0..99=[10..99];
var count, nrcounted:integer;
procedure dispcharset(theset:charset; startch,endch:char);
var indexch:char; count:integer;
begin
count:=1;
for indexch:=startch to endch do
if indexch in theset then
begin
write(indexch:3);
if count mod 20 =0 then
writeln;
inc(count);
end;
end;
begin
nrcounted:=0;
writeln('smallLetters:');
dispcharset(smallletters,'a','z');
writeln;
writeln('doubledigitset:');
for count:=0 to 100 do
begin
if count in doubledigitset then
begin
write(count:3);
inc(nrcounted);
```

```
if nrcounted mod 20=0 then
writeln;
end;
end;
readln;
end.
```

خرج البرنامج أعلاه هو:

```
smallLetters:
  a  c  e  i  m  n  o  r  s  u  v  w  x  z
doubledigitset:
 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
 90 91 92 93 94 95 96 97 98 99
```

يعرف البرنامج السابق ثلاث مجموعات ثابتة الأولى هي smallLetters و قد عرفت على انها مجموعة من نوع المعطيات المسمى charset و الثانية هي singledigitse و الثالثة هي doubledigitset .
ومن أجل تحديد مجموعة عديمة الاسم anonymous set علينا وضع القيم التي نريدها ضمن قوسين مربعين و من ثم يمكننا استخدام هذه المجموعة ضمن التعابير البوليانية التي ينشئها المعامل IN كما في المثال التالي:

```
If userschar  in ['b'..'e','g','p','t','v','z']; then
  . . .
```

مسألة:

لدينا ملفان نصيان info1. Txt و info2.txt وضعت في الملف الأول أرقام طلاب الناجحين في مادة البرمجة1 ووضعت في الملف الثاني أرقام طلاب الناجحين في مادة البرمجة2 علماً أن عدد الطلاب الأعظمي هو 240 طالب في كلتا المادتين(أي أرقام الطلاب هي ضمن **المجال المنفصل** [1..240]) و المطلوب مستفيداً من بنية المعطيات set اكتب برنامج بلغة التربو باسكال يقوم بطباعة أرقام الطلاب الناجحين في المادتين معاً على شاشة الحاسب و ضمن ملف نصي result.txt علما أن كل سكر من أسطر الملفين يحويان رقم طالب معين.
مثال:
لنفرض أن أرقام الطلاب الناجحين في كلتا المادتين هي كالتالي:

Info2.txt	Info1.txt
1	7
8	8
20	9
22	11
23	10
24	22
60	20
	23
	24
	39

عندها يتم طباعة على شاشة الملف و ضمن المف النصي ارقام الطلاب التالية:

8, 20, 22, 23, 24

Code

```
program test;
const m1='c:\info1.txt';
m2='c:\info2.txt';
type
s=set of 1..240;
procedure read_info(var s1:s);
type txt=text;
var f1:text; n:integer;  m:string;
begin
s1:=[];
writeln('please enter the position file');
readln(m);
while(m<>m1)and(m<>m2)do
begin
writeln('please replay enter the position file or prees e to exit');
readln(m);
if(m='e')then exit;
end;
assign(f1,m);
reset(f1);
while(not eof(f1))do
begin
readln(f1,n);
s1:=s1+[n];
end;
close(f1);
end;
procedure print(s1,s2:s);
var s3:s; i:integer;var f:text;
begin
assign(f,'c:\result.txt');
rewrite(f);
i:=1;
s3:=s1*s2;
while(s3<>[])do
begin
if(i in s3)then
begin
write(i:3,', ');
write(f,i:3,', ');
s3:=s3-[i];
if (i mod 15=0)then writeln;
end;
inc(i);
end;
writeln;
writeln(f);
close(f);
end;
```

```
var s1,s2:s;
begin
read_info(s1);
read_info(s2);
writeln('students numbers successful in prog1 and prog2 is: ');
print(s1,s2);
readln;
end.
```

خرج البرنامج السابق بافتراض أن أرقام الطلاب المخزنة في كلا الملفين هي نفسها الأرقام الواردة في مثالنا:

```
please enter the position file
c:\info1.txt
please enter the position file
c:\info2.txt
students numbers successful in prog1 and prog2 is:
  8,  20,  22,  23,  24,
```

مسألة:

اكتب الإجرائيات اللازمة لطباعة وحساب ناتج ضرب مصفوفتين mat1*mat2 علماً أن شرط الضرب هو أن يكون عدد أعمدة المصفوفة الأولى يساوي عدد أسطر المصفوفة الثانية (دون استخدام أي تابع جاهز)

مثال:

$$\begin{bmatrix}1 & 4\\2 & 5\\3 & 6\end{bmatrix} * \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} = \begin{pmatrix}17 & 22 & 27\\22 & 29 & 36\\27 & 36 & 45\end{pmatrix}$$

نلاحظ أن المصفوفة الناتجة تساوي عدد أسطر المصفوفة الأولى * عدد أعمدة المصفوفة الناتجة (قاعدة).

Code

```
program test;
const maxn=10;
type
matrix=array[1..maxn,1..maxn] of integer;
procedure read_info(var mat1,mat2:matrix; n,m,k:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
for j:=1 to m do
begin
write('mat1[',i,'][',j,']= ');
readln(mat1[i][j]);
end;
end;
for i:=1 to m do
begin
for j:=1 to k do
begin
write('mat2[',i,'][',j,']= ');
readln(mat2[i][j]);
end;
end;
end;
```

```
procedure print(mat1,mat2:matrix;n,m,k:integer);
var i,j:integer;
begin
writeln('mat1 is ' );
for i:=1 to n do
begin
for j:=1 to m do
begin
write(mat1[i][j]:4);
end;
writeln;
end;
writeln('mat2 is ');
for i:=1 to m do
begin
for j:=1 to k do
write(mat2[i][j]:4);
writeln;
end;
end;
procedure print1(mat:matrix;n,m:integer);
var i,j:integer;
begin
writeln('mat1 * mat2  is : ');
for i:=1 to n do
begin
for j:=1 to m do
write(mat[i][j]:5);
writeln;
end;
end;
procedure multarray(mat1,mat2:matrix;var mat3:matrix; n,m,k:integer);
var i,j,l:integer;
begin
for i:=1 to n do
for j:=1 to k do
begin
mat3[i,j]:=0;
for l:=1 to m do
mat3[i,j]:=mat3[i,j]+mat1[i,l]*mat2[l,j];
end;
end;
var mat1,mat2,mat3:matrix;
n,m,k:word;
begin
n:=3;
m:=2;
k:=3;
read_info(mat1,mat2,n,m,k);
print(mat1,mat2,n,m,k);
```

```
multarray(mat1,mat2,mat3,n,m,k);
print1(mat3,n,k);
readln;
end.
```

خرج البرنامج السابق هو بفرض أننا أدخلنا عناصر المصفوفتين الوارتين في المثال أعلاه:

```
mat1[1][1]= 1
mat1[1][2]= 4
mat1[2][1]= 2
mat1[2][2]= 5
mat1[3][1]= 3
mat1[3][2]= 6

mat2[1][1]= 1
mat2[1][2]= 2
mat2[1][3]= 3
mat2[2][1]= 4
mat2[2][2]= 5
mat2[2][3]= 6
mat1 is
   1   4
   2   5
   3   6
mat2 is
   1   2   3
   4   5   6
mat1 * mat2  is :
   17   22   27
   22   29   36
   27   36   45
```

مسألة :

المطلوب كتابة الإجرائيات والتوابع اللازمة لقراءة مصفوفة مربعة (عدد الأسطر و الأعمدة في كلا المصفوفتين متساويين=n) و بالتالي عدد عناصر المصفوفة المربعة هو n^2 .

و حساب مجموع كلاً من :

- عناصر كل سطر
- عناصر كل عمود

حتى نحسب مجموع كل سطر نحتاج إلى نسق احادي نضع فيه مجاميع الأسطر و كذلك الأمر لمجاميع الأعمدة.

Code

```
PROGRAM TEST;
const maxn=10;
type
matrix=array[1..maxn] of array[1..maxn] of integer;
mat=array[1..maxn] of integer;
procedure read_info(var m:matrix; n:integer);
var i,j:integer;
begin
for i:=1 to n do
```

```
for j:=1 to n do
begin
write('mat[',i,'][',j,']=  ');
readln(m[i][j]);
end;
end;
procedure sumrows(var sumr:mat; mat:matrix; n:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
sumr[i]:=0;
for j:=1 to n do
sumr[i]:=sumr[i]+mat[i][j];
end;
end;
procedure sumcolumn(var sumc:mat; mat:matrix; n:integer);
var i,j:integer;
begin
for j:=1 to n do
begin
sumc[j]:=0;
for i:=1 to n do
sumc[j]:=sumc[j]+mat[i][j];
end;
end;
procedure printmatrix(mat:matrix; n:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
for j:=1 to n do
write(mat[i][j]:4);
writeln;
end;
end;
procedure print_mat(mat:mat;n:integer);
var i,j:integer;
begin
for i:=1 to n do
writeln('total ',i,' = ',mat[i]:4);
end;
var
n:integer;
mat1:matrix;
sr,sc:mat;
begin
n:=3;
read_info(mat1,n);
sumrows(sr,mat1,n);
```

```
sumcolumn(sc,mat1,n);
writeln('mat is : ');
printmatrix(mat1,n);
writeln('totla lines in mat is: ');
print_mat(sr,n);
writeln('totla columns in mat is: ');
print_mat(sc,n);
readln;
end.
```

خرج البرنامج السابق شبيه بالتالي:

```
mat[1][1]=  2
mat[1][2]=  0
mat[1][3]=  4
mat[2][1]=  1
mat[2][2]=  6
mat[2][3]=  4
mat[3][1]=  8
mat[3][2]=  0
mat[3][3]=  5
mat is :
   2   0   4
   1   6   4
   8   0   5
totla lines in mat is:
total 1 =    6
total 2 =   11
total 3 =   13
totla columns in mat is:
total 1 =   11
total 2 =    6
total 3 =   13
```

مسألة:

أكتب إجرائية بلغة باسكال و سمها Fraction تمرر إليه متحول حقيقي n ويرد قيمة كسرية (بسط و مقام) على نحو **مختزل**.
مثال من أجل n=3.5 يرد الإجراء الكسر المختزل 7/2. ومن أجل n=0.75 يرد الإجراء الكسر المختزل 3/4 .

Code

```
program test;
procedure Fraction(n:real);
var n1,m:real; i,t:integer;
begin
m:=n;
t:=trunc(n);
n1:=n-t;
i:=1;
while(n1<>0)do
begin
i:=i+1;
n:=n*i;
t:=trunc(n);
n1:=n-t;
if(n1<>0)then
n:=m;
```

```
end;
writeln(trunc(n),'/',i);
end;
var n:real;
begin
readln(n);
fraction(n);
readln;
end.
```

<u>الحل بأسلوب آخر:</u>

```
program test;
procedure Fraction(n:real);
var n1:real; i:integer;
begin
n1:=n;
i:=1;
while(frac(n)<>0)do
begin
i:=i+1;
n:=n*i;
if(frac(n)<>0)then
n:=n1;
end;
writeln(trunc(n),'/',i);
end;
var n:real;
begin
readln(n);
fraction(n);
readln;
end.
```

خرج البرنامج السابق من أجل n= 3.5 و n=0.75 و n=98.9 و n=44.8 هو كالتالي:

```
 3.5
7/2

 0.75
3/4

98.9
989/10

44.8
224/5
```

مسألة:
بلغ عدد مشتركي مؤسسة الكهرباء في مدينة معضمية الشام بريف دمشق 1000 مشترك اشتراكاً منزلياً و قد نظمت المعلومات عنهم على بطاقات تسجل عليها المعلومات الآتية:

رقم الاشتراك	اسم المشترك	التأشيرة السابقة للعداد	التأشيرة الحالية للعداد
NB	NAME	TB	TA

حيث يمثل الفرق بين التأشيرة الحالية و التأشيرة السابقة كمية استهلاك المشترك في الدورة كما ان تعرفة الطاقة المعدة للاستهلاك المنزلي في الدورة تخضع لنظام الشرائح التالية وفق الجدول التالي:

من	إلى	الوحدة	سعر - ق. س	
1	100	ك.و.س	25	لكل كيلو واط ساعي
101	200	ك.و.س	35	لكل كيلو واط ساعي
201	400	ك.و.س	50	لكل كيلو واط ساعي
401	600	ك.و.س	75	لكل كيلو واط ساعي
601	فما فوق	ك.و.س	250	لكل كيلو واط ساعي

كما يضاف إلى قيمة الاستهلاك رسم عداد قدره 50 ليرة سورية ورسوم أخرى بمعدل 16% من قيمة الاستهلاك. فإذا علمت أن قيمة الاستهلاك و قيمة الرسوم تقرب إلى أقرب ليرة سورية .
اكتب برنامجاً يعمل على إدخال هذه المعلومات على شكل انساق و يحسب بالنسبة لكل مشترك قيمة الفاتورة المستحقة و يطبع النتائج لجميع المشتركين على النحو التالي:

رقم الاشتراك	تأشيرة سابقة	تأشيرة حالية	كمية استهلاك	قيمة استهلاك	قيمة الرسوم	المبلغ (ل.س)

Code

```
program test;
const maxn=1000; counter=50;
type
subscriber=record
NB,TB,TA:integer;
name:string[25];
value,tax,total:real;
end;
matrix=array[1..maxn] of subscriber;
var qua:integer;
procedure input_inforec(var mat:matrix;n:integer);
var i:integer;
begin
for i:=1 to n do
with mat[i] do
begin
writeln('enter the number of subscriber NO...',i);
readln(nb);
writeln('enter the name of subscriber No...',i);
readln(name);
writeln('enter the previous reading for subscriber...',i);
readln(tb);
writeln('enter the current reading for subscriber...',i);
readln(ta);
end;
end;
```

```
var X:matrix; n,i:integer;
begin
n:=4;
input_inforec(x,n);
writeln('nb':6,' ','name':15,' ',
'tb':6,' ','ta':6,' ','quantity':6,' ','value':8,' ','tax':4,' ','counter':5,' ','total(S.P)':8);
for i:=1 to n do
with x[i] do
begin
qua:=ta-tb;
case Qua of
0..100:value:=0.25*Qua;
101..200:value:=25+0.35*(Qua-100);
201..400:value:=60+0.50*(Qua-200);
401..600:value:=160+0.75*(Qua-400);  {160=0.25*100+0.35*100+0.5*200}
else
value:=310+2.5*(Qua-600); {310=0.25*100+0.35*100+0.5*200+0.75*200}
end;
value:=round(value);
tax:=round(0.16*value);
total:=value+tax+counter;
writeln(nb:6,' ',name:15,' ',
tb:6,' ',ta:6,' ',qua:6,' ',value:8:0,' ',tax:5:0,' ',counter:5,' ',total:10:0);
end;
readln;
end.
```

خرج البرنامج السابق هو:

```
enter the number of subscriber NO...1
1
enter the name of subscriber No...1
kgaled yassin
enter the previous reading for subscriber...1
627
enter the current reading for subscriber...1
1488
enter the number of subscriber NO...2
292
enter the name of subscriber No...2
ali omar
enter the previous reading for subscriber...2
16817
enter the current reading for subscriber...2
16876
enter the number of subscriber NO...3
476
enter the name of subscriber No...3
omar yousef
enter the previous reading for subscriber...3
6589
enter the current reading for subscriber...3
7255
enter the number of subscriber NO...4
563
enter the name of subscriber No...4
iman alsheikh
enter the previous reading for subscriber...4
13621
enter the current reading for subscriber...4
14213
    nb           name     tb     ta quantity    value  tax counter total(S.P)
     1  kgaled yassin    627   1488      861      963  154      50      1167
   292       ali omar  16817  16876       59       15    2      50        67
   476    omar yousef   6589   7255      666      475   76      50       601
   563  iman alsheikh  13621  14213      592      304   49      50       403
```

شرح البرنامج السابق هو:
يمثل الثابت counter رسم العداد و يساوي 50 lira و الثابت maxn يمثل العدد الأقصى للمشتركين subscribers و لدينا السجل subscriber الذي يمثل بيانات عن المشترك حيث يمثل الحقل nb رقم المشترك (لا يتكرر) و الحقل name يمثل اسم المشترك و الحقل Tb يمثل التأشيرة السابقة للعداد و الحقل Ta يمثل التأشيرة الحالية للعداد و الحقل value يمثل قيمة الاستهلاك حسب نظام الشرائح و يقرب إلى أقرب ليرة سورية و الحقل Tax يمثل الرسوم (من قيمة الاستهلاك value) و نسبة الرسوم هي 16% و تقرب إلى اقرب ليرة سورية أما الحقل total فيمثل قيمة الفاتورة المستحقة و هو عبارة عن جمع الحقل value+tax+counter .
أما المتغير Qua فيمثل كمية الاستهلاك.

مسألة:
المطلوب حل المسألة السابقة باستخدام السلاسل الخطية (المؤشرات) وحيدة الارتباط . (ما هي ميزة الحل هذا عن سابقه).

مسألة :

يعمل في إحدى المؤسسات الصناعية بمدينة معضمية الشام 100 عامل و قد نظمت المؤسسة المعلومات المتعلقة بكل منهم على بطاقة تشتمل على حقول لرقم العامل و اسمه الكامل و عدد ساعات عمله الأسبوعية و الأجر الساعي كما تركت حقولاً لتدوين عدد ساعات العمل الإضافي و أجر ساعة العمل الإضافي و الأجر الإجمالي وضريبة الدخل المستحقة وضريبة الدخل المستحقة على كل عامل فإذا علمت أن العامل يتقاضى أجراً إضافياً عن ساعات العمل الأسبوعية التي تزيد عن 35 ساعة أسبوعية بمعدل 1.5 أجر ساعة العمل العادية كما يخضع الأجر الأسبوعي إلى ضريبة دخل بمعدل 8% إذا كان أقل من 7000 ليرة و 11% في الحالات الأخرى.

اكتب برنامجاً يعمل على إنشاء سجل يشتمل على كافة هذه المعلومات ثم يحسب الأجر الصافي المستحق للعامل أسبوعياً كما يعمل على استدعاء إجرائية لترتيب هؤلاء العمال تصاعدياً وفقاً لأرقامهم و يطبع النتائج.

Code

```
program test;
const maxn=100;
type
worker=record
nb:integer;
name:string[25];
Hrs,rate,Ext,Orate,T,Tax,Net:real;
end;
matWork=array[1..maxn] of worker;
var X:matwork;

procedure input_inforec(var mat:matwork;n:integer);
var i:integer;
begin
for i:=1 to n do
begin
with mat[i] do
begin
writeln('enter the number of worker...',i);
readln(nb);
writeln('enter the name of worker...',i);
readln(name);
writeln('enter weekly working hours of worker...',i);
readln(Hrs);
writeln('enter Hourly rate for worker...',i);
readln(rate);
Orate:=1.5*rate;
if hrs>35 then
begin
Ext:=Hrs-35;
T:=35*rate+Ext*Orate;
end
else
begin
Ext:=0;
T:=Hrs*rate;
end;
if T<7000 then
Tax:=0.08*T
else
Tax:=0.11*T;
```

```
Net:=T-Tax;
end;
end;
end;

procedure sort(var mat:matwork; n:integer);
var i,j:integer; W:worker;
begin
for i:=1 to n-1 do
for j:=i+1 to n do
if(mat[i].nb>mat[j].nb)then
begin
W:=mat[i];
mat[i]:=mat[j];
mat[j]:=W;
end;
end;
procedure print(mat:matwork; n:integer);
var i:integer;
begin
writeln('Nb':6,'name':15,' rate':8,' ext':8,' Orate':8,' T':10,' Tax':9,' net':10);
for i:=1 to n do
with mat[i] do
writeln(Nb:6,name:15,rate:8:2,ext:8:1,' ',Orate:8:1,' ',
T:12:2,' ',Tax:8:2,'  ',trunc(net));
end;
var n:integer;
begin
n:=4;
input_inforec(X,n);
sort(X,n);
print(X,n);
readln;
end.
```

خرج البرنامج السابق:

```
enter the number of worker...1
1
enter the name of worker...1
khaled yassin
enter weekly working hours of worker...1
 40
enter Hourly rate for worker...1
 160
enter the number of worker...2
105
enter the name of worker...2
soliman alsheikh
enter weekly working hours of worker...2
 35
enter Hourly rate for worker...2
 150
enter the number of worker...3
3
enter the name of worker...3
ali omar
enter weekly working hours of worker...3
 39
enter Hourly rate for worker...3
 140
enter the number of worker...4
2
enter the name of worker...4
Iman omar
enter weekly working hours of worker...4
 41
enter Hourly rate for worker...4
 155
    Nb              name    rate     ext   Orate          T       Tax     net
     1     khaled yassin  160.00     5.0    240.0    6800.00    544.00   6256
     2         Iman omar  155.00     6.0    232.5    6820.00    545.60   6274
     3          ali omar  140.00     4.0    210.0    5740.00    459.20   5280
   105  soliman alsheikh  150.00     0.0    225.0    5250.00    420.00   4830
```

شرح البرنامج السابق :

يمثل السجل worker حقول المعطيات التي تمثل معلومات عن العامل من حيث الاسم و الرقم و الأجر حيث يمثل الحقل NB رقم العامل (لا يتكرر) و الحقل name اسم العامل على 25 خانة كحد أقصى و الحقل HRS يمثل عدد ساعات العمل الأسبوعية للعامل و الحقل rate يمثل أجر الساعي للعامل و الحقل Ext يمثل عدد الساعات الإضافية (التي تزيد عن 35 ساعة) و الحقل Orate يمثل أجر ساعة العمل الإضافية و الحقل T الأجر الإجمالي و الحقل TAX يمثل ضريبة الدخل و الحقل net يمثل الأجر الصافي المستحق للعامل أسبوعياً.

و حيث الإجرائية input_inforec مهمتها إدخال البيانات لمعالجتها لتصبح معلومات مفيدة . و الإجرائية sort تقوم بعملية فرز للمعلومات في مصفوفة سجل العمال المخزنة ضمنها حسب رقم العامل (الفرز بالتعويم أو الفرز الفقاعي) و الإجرائية print تقوم بطباعة المعلومات الخاصة بكل عامل غلى شاشة الخرج القياسي "الشاشة".

مسألة:

اكتب إجرئيتان لطباعة عناصر سلسلة (مؤشرات) وحيدة الارتباط بأسلوبين مختلفان نوعا ما باعتبار وجود حقل معطيات واحد val.

Code

```
Procedure print1(var p:plist);
Begin
If p<>nil then
begin
Write(p^.val,',  ');
Print1(p^.next);
End;
End;
```

الأسلوب الآخر (الطباعة العكسية):

```
Procedure print2(var p:plist);
Begin
If p<>nil then
begin
Print2(p^.next);
Write(p^.val,',  ');
End;
End;
```

أسلوب آخر أيضاً:

```
Procedure print;
Var i:integer;
Begin
For i:=1 to length(head)do
Begin
Write(p^.val);
P:=p^.next;
End;
End;
```

ما هو خرج البرنامج التالي:

Code

```
program test;
const n=4;
var x:array[1..4] of array[1..4] of integer;
i,j:integer; Z,y:real;
b:boolean;
begin
y:=5.5;
z:=7.0;
b:=not(y<>z) and(z>(z*y));
writeln('y=',y,' ','z=',z:3);
writeln;
write('B=',b);
if(b=true)then
for i:=1 to n do
begin
```

```
for j:=1 to n do X[i,j]:=j*2;
end
else
for i:=1 to n do
begin
for j:=1 to n do
begin
if i<>j then
X[i,j]:=i*j
else
x[i,j]:=0;
end;
end;
for i:=1 to n do
begin
for j:=1 to n do
write(x[i,j]:4);
writeln;
end;
readln
end.
```

C	B	A
y= 5.5000000000E+00 z= 7.0E+00 B=TRUE 2 4 6 8 2 4 6 8 2 4 6 8 2 4 6 8	y= 5.5000000000E+00 z= 7.0E+00 B=FALSE 0 2 3 4 2 0 6 8 3 6 0 12 4 8 12 0	y= 5.5000000000E+00 z= 7.0E+00 B=FALSE 0 2 3 4 2 0 6 8 3 6 0 12 4 8 12 0
	E	D
	الجواب الصحيح ليس مما سبق	y= 5.5000000000E+00 z= 7.0E+00 B=TRUE 2 4 6 8 2 4 6 8 2 4 6 8 2 4 6 8

جامعة دمشق كلية الهندسة المعلوماتية المهندس خالد ياسين الشيخ 2011	امتحان البرمجة1 عدد الأسئلة 55 سؤال فقط. خرج البرامج من اليسار لليمين

ر.س	البرامج Programs	الخيارات Options
1	PROGRAM TEST; type int=^integer; procedure ff(const m:integer); begin write(m,','); end; procedure f(p:int); begin p^:=p^+88; end; var m:integer; begin m:=4; ff(m); f(@m); writeln(m); readln; end.	ما هو خرج البرنامج المبين يسارا: a. **4,92** b. 4,88 c. 99,98 d. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه. e. الجواب الصحيح ليس مما سبق.
2	PROGRAM TEST; type m=array[1..5] of integer; const matrix:m=(5,3,4,2,1); var i:integer; p:^integer; begin i:=9; p:=@i; i:=i+1; writeln(p^); readln; end.	حدد الجواب الصحيح من أجل البرنامج المبين يساراً: a. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه. b. خرج البرنامج هو 9. c. **خرج البرنامج هو 10.** d. تتم طباعة قيمة عشوائية . e. كافة الإجابات السابقة خاطئة.
3	PROGRAM TEST; Type m=array[1..3,1..2]of integer; const matrix:m=((1,2),(3,4),(5,6)); var p:^integer;i:integer; begin i:=99; p:=@i; p:=@matrix; writeln(p^); readln end.	حدد الجواب الصحيح من أجل البرنامج المبين يساراً: a. خرج البرنامج هو 99 b. خرج البرنامج هو 4 c. **خرج البرنامج هو 1** d. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه. e. الجواب الصحيح ليس مما سبق.

4	PROGRAM TEST; var i,s,n:integer; begin s:=0; n:=4; for i:=1 to n do s:=s+(sqr(i) div i); write(s); readln; end.	حدد الجواب الصحيح من أجل البرنامج المبين يساراً: a. البرنامج يقوم بحساب مجموع الأعداد من 1 إلى n^2. b. البرنامج يقوم بحساب مجموع الأعداد من 1 إلى n/2. **c. البرنامج يقوم بحساب مجموع الأعداد من 1 إلى n.** d. البرنامج يقوم بحساب مجموع الأعداد من 1 إلى n/i. e. الجواب الصحيح ليس مما سبق.
5	PROGRAM TEST; var i,s,n:longint; begin s:=1; n:=8; for i:=1 to n do s:=s*(sqr(i) div i); write(s); readln; end.	حدد الجواب الصحيح من أجل البرنامج المبين يساراً: a. البرنامج يقوم بحساب مجموع الأعداد من 1 إلى n^2. b. البرنامج يقوم بحساب مضروب الأعداد من 1 إلى n^2. **c. البرنامج يقوم بحساب مضروب الأعداد من 1 إلى n.** d. البرنامج يقوم بحساب مضروب الأعداد من 1 إلى n/i e. جميع الإجابات السابقة خاطئة.
6	PROGRAM TEST; function f(const n:integer):integer; begin if(n=0)then f:=1 else f:=n div 2+1+f(n-1) end; var n:integer; begin n:=2; n:=f(f(f(n))); writeln(n); readln; end.	ما هو خرج البرنامج المبين يساراً: a. 25 **b. 30** c. 9 d. 4 e. الجواب الصحيح ليس مما سبق.
7	PROGRAM TEST; function f(const n:integer):integer; begin if(n<=0)then f:=n else f:=n div 2+1+f(n-2*3); end; var n:integer; begin n:=9; n:=n+5; n:=f(f(f(n))); writeln(n); readln; end.	ما هو خرج البرنامج المبين يساراً: a. 11 b. 8 **c. 3** d. 4 e. None of The above

8

```
PROGRAM TEST;
function f(const y,c:integer):integer;
begin
if(c=0)then
f:=y
else
f:=f(y+c div 2,c-1);
end;
var n:integer;
begin
n:=9;
n:=n+5;
n:=f(f(n,4),4);
writeln(n);
readln;
end.
```

ما هو خرج البرنامج المبين يساراً:

a. 11
b. 8
c. <u>22</u>
d. 18
e. None of The above

9

```
PROGRAM TEST;
type
matrix=array [1..5] of integer;
var mat1:matrix;i,n:integer;
procedure strange( mat:matrix;size:integer);
var i,j,temp:integer;
begin
for i:=size downto 1 do
for j:=2 to i do
if(mat[j-1]>mat[j])then
begin
temp:=mat[j-1];
mat[j-1]:=mat[j];
mat[j]:=temp;
end;
end;
procedure print(mat:matrix;n:integer);
var i:integer;
begin
for i:=1 to n do
write(mat[i],',');
writeln;
end;
begin
n:=5;
for i:=1 to n do
readln(mat1[i]);{ (77,88,99,5,9)}
strange(mat1,n);
print(mat1,n);
readln;
end.
```

بفرض لدينا محتويات النسق MAT1:

	1	2	3	4	5
MAT1	77	88	99	5	9

تصبح محتويات النسق بعد التنفيذ البرنامج المبين يساراً:

a. محتويات النسق كالتالي:

	1	2	3	4	5
MAT1	9	5	99	88	77

b. محتويات النسق كالتالي:

	1	2	3	4	5
MAT1	5	9	77	88	99

c. محتويات النسق كالتالي:

	1	2	3	4	5
MAT1	99	88	77	9	5

d. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه.

e. <u>الجواب الصحيح ليس مما سبق.</u>

10	PROGRAM TEST; function func(const n:integer):integer; var i,sum:integer; begin i:=-2; sum:=i; repeat i:=i+1; sum:=sum+i; until(n=i); func:=sum+i div 2; end; const max=5; var n:integer; begin n:=max; n:=func(n); writeln(n); readln end.	ما هو خرج البرنامج المبين أدناه: a. 17 b. 15 c. 12 **d. <u>14</u>** e. الجواب الصحيح ليس مما سبق.
11	PROGRAM TEST; function func(const n:integer):integer; var i,sum:integer; begin i:=-2; sum:=i; n:=2; repeat i:=i+1; sum:=sum+i; until(n=i); func:=sum; end; const max=5; var n:integer; begin n:=max; n:=func(n); writeln(n); readln end.	ما هو خرج البرنامج المبين يساراً: a. 0 b. 1 c. 12 d. 11 **e. <u>الجواب الصحيح ليس مما سبق.</u>**
12	PROGRAM TEST; procedure f1(n, res:integer); begin if(n>=0)then begin res:=res+n; f1(n-2,res); res:=res+n; end; end; var s:integer; begin s:=10; f1(5,s); writeln(s); readln end.	ما هو خرج البرنامج المبين يساراً: a. 27 b. 28 c. 25 **d. <u>10</u>** e. الجواب الصحيح ليس مما سبق.

13	PROGRAM TEST; procedure f1(n:integer; var res:integer); begin if(n>=0)then begin res:=res+n; f1(n-2,res); res:=res+n; end; end; var s:integer; begin s:=10; f1(5,s); writeln(s); readln end.	ما هو خرج البرنامج المبين يساراً: a. 18 b. 17 c. 19 **d. 28** e. الجواب الصحيح ليس مما سبق.
14	PROGRAM TEST; type monthe=(jan,feb,mar,apr,may,june); var mon:monthe; begin mon:=feb; case ord(mon) of 0..1:writeln(ord(mar)); 2..7: writeln(ord(june)); else writeln(ord(apr)); end; readln end.	ما هو خرج البرنامج المبين يساراً: a. 0 **b. 2** c. 3 d. 5 e. الجواب الصحيح ليس مما سبق.
15	PROGRAM TEST; var a,b:integer; begin a:=9; b:=88; a:=a xor b; b:=a xor b; a:=a xor b; writeln('a=',a,',b=',b); readln end.	ما هو خرج البرنامج المبين يساراً: a. a=9,b=88 **b. a=88,b=9** c. a=81,b=9 d. a=81,b=88 e. الجواب الصحيح ليس مما سبق.
16	PROGRAM TEST; var a,b:integer; begin a:=9; b:=88; a:=a or b; b:=a or b; a:=a and b; writeln('a=',a,',b=',b); readln end.	ما هو خرج البرنامج المبين يساراً: a. a=89,b=88 b. a=200,b=88 c. a=88,b=88 d. a=89,b=88 **e. الجواب الصحيح ليس مما سبق.**

17	PROGRAM TEST; var a,b:integer; begin randomize; a:=10; b:=a; while(a<>0)do begin write(random(b),','); a:=a-1; end; readln; end.	حدد الجواب الصحيح من أجل البرنامج المبين يساراً: a. تتم طباعة عشرة قيم صحيحة عشوائية نفسها في كل مرة ننفذ فيها البرنامج b. تتم طباعة عشرة قيم صحيحة عشوائية مختلفة في كل مرة ننفذ فيها البرنامج c. تتم طباعة قيم صحيحة ضمن المجال المغلق من [0..9] فقط. **d. <u>b و c صحيحتان</u>** e. a و c صحيحتان.
18	PROGRAM TEST; type matrix=array[1..5] of integer; procedure strange(var mat:matrix;n:integer); var temp,i:integer; begin for i:=1 to n div 2 do begin temp:=mat[n-i+1]; mat[n-i+1]:=mat[i]; mat[i]:=temp; end; end; procedure print(mat:matrix;n:integer); var i:integer; begin for i:=1 to n do write(mat[i],','); writeln; end; var mat1:matrix; begin mat1[1]:=3; mat1[mat1[1]]:=2; mat1[mat1[3]]:=4; mat1[mat1[2]]:=5+mat1[mat1[3]]; mat1[5]:=mat1[1]+3; strange(mat1,5); print(mat1,5); readln end.	ما هو خرج البرنامج المبين يساراً: a. 6,9,4,2,1, b. 6,9,2,4,6 **c. <u>6,9,2,4,3,</u>** d. 3,4,2,6,9, e. الجواب الصحيح ليس مما سبق.

19	type matrix=array[1..3,1..3] of integer; function strange(mat:matrix;n:integer):integer; var i,j,res:integer; begin res:=0; for i:=1 to n do for j:=1 to i do res:=res+mat[i,j]; strange:=res end;	بفرض لدينا مصفوفة mat مربعة 3×3 محتوياتها كالتالي: $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 9 \\ 3 & 2 & 5 \end{pmatrix}$ و المطلوب ما هي القيمة التي يردها التابع strange المبين يساراً: a. 13 b. 7 c. 16 **d. 14** e. الجواب الصحيح ليس مما سبق.
20	type matrix=array[1..3,1..3] of integer; function strange(mat:matrix;n:integer):integer; var i,j,res:integer; begin res:=0; for i:=n downto 1 do for j:=n downto i do res:=res+mat[i,j]; strange:=res end;	بفرض لدينا مصفوفة mat مربعة 3×3 محتوياتها كالتالي: $\begin{pmatrix} 1 & 2 & 3 \\ 1 & 12 & 2 \\ 3 & 4 & 5 \end{pmatrix}$ و المطلوب ما هي القيمة التي يردها التابع strange المبين يساراً: a. 8 b. 26 c. 7 **d. 25** e. الجواب الصحيح ليس مما سبق.
21	type matrix=array[1..4,1..4] of integer; function strange(mat:matrix;n:integer):integer; var i,j,res:integer; begin res:=0; for i:=n downto 1 do for j:=3 downto 1 do res:=res+mat[i,j]; strange:=res end;	بفرض لدينا مصفوفة mat مربعة 4×4 محتوياتها كالتالي: $\begin{Bmatrix} 6 & 1 & 2 & 1 \\ 3 & 1 & 4 & 2 \\ 8 & 2 & 3 & 2 \\ 0 & 1 & 0 & 2 \end{Bmatrix}$ و المطلوب ما هي القيمة التي يردها التابع strange المبين يساراً: a. 36 b. 30 **c. 31** d. 32 e. الجواب الصحيح ليس مما سبق.
22	PROGRAM TEST; procedure ff(n:integer;var res:integer); begin if(n<>0)then begin res:=res+n; ff(n-1,res); res:=res+n; end; end; var res:integer; begin res:=0; ff(4,res); writeln(res); readln end.	ما هو خرج البرنامج المبين يساراً: a. 0 **b. 20** c. 10 d. 25 e. الجواب الصحيح ليس مما سبق.

23	PROGRAM TEST; function ff(n:longint):integer; begin if(n<>0)then ff:=ff(n div 10)+1 else ff:=n; end; var res:longint; begin res:=33012; res:=ff(ff(res)); write(res); readln end.	ما هو خرج البرنامج المبين يساراً: a. 5 b. 9 **c. 1** d. 0 e. الجواب الصحيح ليس مما سبق.
24	PROGRAM TEST; var n:integer; begin n:=1; while(true) do begin if(5>=n) then begin write(n,','); if(n mod 5=0)then break; n:=n+1; end; end; readln end.	ما هو خرج البرنامج المبين يساراً: a. يطبع قيمة n خمس مرات و يدخل في حلقة غير منتهية. b. يطبع قيمة n أربع مرات و يدخل حلقة غير منتهية. **c. يطبع قيمة n خمس مرات و يخرج من الحلقة.** d. يطبع قيمة n أربع مرات و يخرج من الحلقة. e. كافة الإجابات السابقة خاطئة.
25	PROGRAM TEST; var ok:boolean; begin readln(ok)(*input ok is true*) if(ok)then write(4>2,',',3<2); readln end.	بفرض لدينا ok متحول Boolean و نريد إعطاءه قيمة true من لوحة الملامس يكون الخرج من أجل البرنامج المبين يساراً هو: a. لا يمكن قراءة أو طباعة متحول من نمط Boolean. **b. يمكن طباعة متحول من نمط Boolean و لا يمكن قراءته** c. خرج البرنامج هو TRUE,TRUE d. خرج البرنامج هو TRUE,FALSE e. الجواب الصحيح ليس مما سبق.
26	PROGRAM TEST; var n,res:integer; c:char; s:string; begin s:=''; s:=s+'syria arabic'; n:=96; n:=succ(n); res:=n; c:='E'; write(s,',',ord(c),',',chr(res),',',n); end.	ما هو خرج البرنامج من أجل البرنامج المبين يساراً: a. syria arabic,65,b,98 **b. syria arabic,69,a,97** c. syria arabic,65,b,97 d. syria arabic,66,a,98 e. الجواب الصحيح ليس مما سبق.

27	PROGRAM TEST; function f(var n:integer):integer; begin n:=n*(n+1) div 2; f:=n; end; var n:integer; begin n:=9; n:=f(5); writeln(n); readln end.	ما هو خرج البرنامج المبين يساراً: a. 9 b. 12 c. 14 d. 15 e. **الجواب الصحيح ليس مما سبق.**
28	اختر العبارة الصحيحة: a. خرج التعليمة التالية ;(write(4mod 9 هو 4. b. يمكن في لغة باسكال دوما استبدال حلقة for بحلقة while و لكن العكس غير صحيح تماماً. c. عند ما نقوم بحجز نسق بلغة باسكال على الشكل ;mat:array [1..5] of integer فإن النسق mat يحجز مساحة مقدرة بــ 10bytes. d. **جميع الإجابات السابقة صحيحة.** e. a و c صحيح.	
29	اختر العبارة الصحيحة: a. الحل التكراري دائماً أفضل من الحل العودي. b. الحل العودي دائماً أفضل من الحل التكراري. c. الحل العودي يستهلك قدرا أقل من الذاكرة مقارنة بالحل التكراري. d. يمكن تحويل جميع الخوارزميات من الحل العودي إلى الحل التكراري دوماً. e. **كل ما ذكر غير صحيح.**	
30	اختر العبارة الصحيحة: a. القيمة nil و dispose متكافئتان. b. يمكن تطبيق تعليمة dispose على مؤشر لم نجري عليه عملية new. c. تأخذ جميع المتحولات بلغة باسكال تلقائياً القيمة صفر باستثناء المتحولات المحلية الخاصة بإجراء ما. d. لا يستطيع مترجم باسكال التعامل مع مكتبات جاهزة أو مضمنة. e. **كل ما ذكر غير صحيح.**	
31	اختر العبارة الخاطئة: a. التابع function بلغة باسكال يجب حتماً أن يعيد قيمة و في حال عدم إسناد قيمة للتابع فإنه يعيد قيمة عشوائية. b. **الإجرائية procedure بلغة باسكال يجب حتماً أن تعيد قيمة كمتحول يدعى متحول دخلخرج.** c. السجل record يمكن أن يحوي أنماط معطيات مختلفة أو متماثلة. d. جميع الإجابات السابقة خاطئة. e. a و b خاطئان تماماً.	
32	اختر العبارة الصحيحة: a. يمكن بلغة باسكال كتابة تابع ضمن تابع آخر. b. يمكن بلغة باسكال استدعاء تابع ضمن تابع آخر. c. يمكن بلغة باسكال كتابة تابع أو إجرائية ضمن جسم البرنامج الرئيسي. d. لغة باسكال لغة مرنة . e. **a و b**	
33	اختر العبارة الصحيحة: a. التعبيران التاليين متكافئان: ((if(n>=10) ≡ if(not(n<10 b. خرج التعليمة التالية: (write(100+2 div 2 هو 51. c. عكس الدالة xor هو xor. d. كل ما سبق صحيح. e. **a و c صحيح.**	

34		اختر العبارة الصحيحة: a. ناتج تنفيذ التعليمة التالية: 51 and 255 هو 51. b. التعبيران التاليين متكافئان: if(not(n<1)) ≡ if(not(not(n>1))) c. عند التعامل مع العمليات الرياضية فإن الأقواس لها الأفضلية على بقية المعاملات. d. كل ما سبق صحيح. e. **a و b صحيح.**
35	PROGRAM TEST; function f(n:integer):boolean; var i:integer;ok:boolean; begin ok:=true; if(n<=1)then ok:=false else for i:=2 to n div 2 do if(n mod i=0)then begin ok:=false; break; end; f:=ok; end; var n,i:integer; begin readln(n); for i:=1 to n do if(f(i))then write(i,','); readln; end.	ما العمل الذي يقوم به البرنامج المبين يساراً: a. **حساب وطباعة الأعداد الأولية ضمن المجال المغلق [1..n].** b. حساب وطباعة الأعداد غير الأولية ضمن المجال المغلق [1..n]. c. حساب وطباعة الأعداد الزوجية ضمن المجال المغلق [1..n]. d. حساب وطباعة الأعداد الفردية ضمن المجال المغلق [1..n]. e. الجواب الصحيح ليس مما سبق.
36	PROGRAM TEST; var n:integer; begin n:=10; if (n mod 10+1=1)then begin while(n<>0)do n:=n-1; writeln(n); end; readln; end.	ما هو خرج البرنامج التالي: a. **0** b. 9 c. 10 d. 5 e. الجواب الصحيح ليس مما سبق.

37	program test; var s1,s2:set of char;c:char; begin s1:=['a'..'z']; s2:=s1-(s1-s1); for c:='a' to 'z' do if c in s2 then write(c,', '); readln; end.	خرج البرنامج المبين يساراً هو: a. a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, b. a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, q, r, c. a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, d. لا يتم طباعة أي شيء على شاشة الحاسب. **e. الجواب الصحيح ليس مما سبق.**
38	program test; var c,n,i:integer; begin n:=5; c:=0; n:=n+2 div 3 mod 5; c:=c-(n-2 div 2 +n mod 2); if (c mod 2=1) and (n mod 4=0)then write(c,',') else write(n,','); for i:=c to n do write(i,', '); readln; end.	خرج البرنامج المبين يساراً هو: a. 0, b. 1, c. 5, d. 1,0, **e. الجواب الصحيح ليس مما سبق.**
39	program test; procedure proce(var c:integer; n:integer); begin c:=c*n div 5; n:=c+20 div 4; end; function func(n,m:integer):integer; begin proce(n,m); func:=n+m; end; var n:integer; begin n:=func(20,9); writeln(n); readln; end.	خرج البرنامج المبين يساراً: a. 77 **b. 45** c. 36 d. 59 e. None of the above

40	program test; procedure proce(var n:integer; m:integer); begin if m<1 then n:=0 else begin proce(n,m-1); proce(n,m-2); n:=n+m; end; end; var m:integer; begin m:=200; proce(m,4); writeln(m); readln; end.	خرج البرنامج المبين يساراً هو: a. 4 **b. <u>6</u>** c. 8 d. 12 e. 206 ملاحظة: هنا كل استدعاء عودي ينسخ متحولاته الخاصة به ضمن الذاكرة الرئيسية.
41	PROGRAM TEST; procedure func(var x,y:integer); begin if(x=0)then y:=1 else begin func(y-1,x); func(x-1,y); y:=y+x; end; end; var x,y:integer; begin x:=3; y:=2; func(x,y); writeln(x,',',y); readln; end.	خرج البرنامج المبين يساراً هو: a. 13,21 b. 16,13 c. 15,12 d. 11,11 **e. <u>الجواب الصحيح ليس مما سبق.</u>**
42	PROGRAM TEST; function sum(x,y:integer):integer; begin if(y=0)then sum:=x else begin sum:=sum(x,y-1); sum:=y+x+sum(y-1,y-1); end; end;	خرج البرنامج المبين يساراً هو: a. 13 b. 16 c. 15 **d. <u>11</u>** e. الجواب الصحيح ليس مما سبق.

	begin writeln(sum(2,3)); readln; end.	
43	PROGRAM TEST; var i:integer; begin i:=0; while(true)do begin i:=i+1; if (i mod 4=0) and (i div 9=1)then break; i:=i+2; end; writeln(i); readln; end.	خرج البرنامج المبين يساراً هو: **a. <u>16</u>** b. 18 c. 10 d. 12 e. الجواب الصحيح ليس مما سبق.
44	PROGRAM TEST; function func(x,y:integer):integer; begin if(x mod y=0)then func:=y else func:=func(y,x mod y); end; begin writeln(func(50,47)); readln; end.	عمل البرنامج المبين يساراً هو: **a. <u>إيجاد القاسم المشترك الأعظم لعددين صحيحين</u>** b. إيجاد المضاعف المشترك الأصغر لعددين صحيحين. c. إيجاد باقي قسمة العدد الصحيح x على العدد الصحيح y. d. إيجاد ناتج القسمة الصحيح للعدد x على العدد الصحيح y. e. الجواب الصحيح يختلف عما سبق.
45	PROGRAM TEST; function fib(n:integer):integer; begin if (n<=0)then fib:=n else if(n=1) or (n=2) then fib:=1 else fib:=fib(n-1)+fib(n-2); end; begin writeln(fib(8)); readln; end.	خرج البرنامج المبين يساراً هو: a. 13 b. 8 c. 34 d. 5 **e. <u>الجواب الصحيح يختلف عما سبق</u>**

<table>
<tr><td>46</td><td><pre>PROGRAM TEST;
function func1(var a:integer):integer;
var x,y:integer;
begin
x:=20 div 2;
y:=x div 2;
a:=x+y;
func1:=a;
end;
function func2(x:integer):integer;
var y:integer;
begin
y:=90 mod 9;
func1(y);
func1(x);
func2:=x+y;
end;
begin
writeln(func2(3115));
readln;
end.</pre></td><td>خرج البرنامج المبين يساراً هو:
a. 3115
b. <u>30</u>
c. 15
d. 31
e. الجواب الصحيح ليس مما سبق.</td></tr>
<tr><td>47</td><td><pre>PROGRAM TEST;
var x,y:integer;
begin
x:=7;
y:=6;
if(x or y and 0=7) then
writeln(y   xor   x)
else
writeln(x);
readln
end.</pre></td><td>خرج البرنامج المبين يساراً هو:
a. 7
b. 6
c. <u>1</u>
d. 0
e. الجواب الصحيح ليس مما سبق.</td></tr>
<tr><td>48</td><td><pre>PROGRAM TEST;
var x,y,z:integer;
begin
x:=7;
y:=6;
z:=x+y mod 2;
if((y>z) or (x>z)) then
writeln(y or x)
else
writeln(x xor z);
readln
end.</pre></td><td>خرج البرنامج المبين يساراً هو:
a. 7
b. 6
c. 1
d. <u>0</u>
e. الجواب الصحيح ليس مما سبق.</td></tr>
</table>

49	PROGRAM TEST; var x,y,n:integer; begin y:=0; n:=20; for x:=1 to n div 6 do y:=y+6*x; writeln(y); readln; end.	عمل البرنامج المبين يساراً هو: a. إيجاد مجموع الأعداد الصحيحة و التي تقبل القسمة على العدد 6 ضمن المجال المغلق [20,0]. b. إيجاد مجموع الأعداد الصحيحة و التي تقبل القسمة على العددين 2 و 3 معاً ضمن المجال المغلق [20,0]. c. إيجاد مجموع الأعداد الصحيحة و التي تقبل القسمة على العدد 2 ضمن المجال المغلق [20,0]. **d. a و b.** e. الجواب الصحيح ليس مما سبق.
50	PROGRAM TEST; var x,y,n,i:integer; begin y:=0; n:=400 div 20; i:=0; repeat i:=i+1; y:=y+6*i until (i=n div 6); writeln(y); readln; end.	خرج البرنامج المبين يساراً هو: a. لا يمكن ترجمة البرنامج و لا يمكن تنفيذه. b. 18 **c. 36** d. 38 e. الجواب الصحيح ليس مما سبق.
51	program test; var i:integer; begin randomize; repeat i:=random(5); until(i mod 10=0); writeln(i); readln; end.	خرج البرنامج المبين يساراً هو: **a. 0** b. لا يمكن معرفة خرج البرنامج. c. 4 d. تسبب حلقة Repeat الدخول بحلقة لا نهائية. e. الجواب الصحيح ليس مما سبق.
52	اختر العبارة الخاطئة: a. إن استخدام حلقة التكرار while يشير إلى اننا نعرف عدد مرات التكرار قبل بداية عمل هذه الحلقة. b. إن استخدام حلقة التكرار for يشير إلى اننا نعرف عدد مرات التكرار قبل بداية عمل هذه الحلقة. c. إن استخدام حلقة التكرار repeat يشير إلى اننا نعرف عدد مرات التكرار قبل بداية عمل هذه الحلقة. **d. جميع الإجابات السابقة صحيحة.** e. a . و c. صحيح.	
53	اختر العبارة الصحيحة: a. إن التخصيص الديناميكي في حجز حجر ضمن الذاكرة الرئيسية لا تتم أثناء ترجمة البرنامج. b. أن التخصيص الساكن في حجز حجر ضمن الذاكرة الرئيسية لا تتم أثناء تنفيذ البرنامج. c. الكومة heap هي المساحة من الذاكرة الرئيسية المتبقية دون استخدام. وتكون مخصصة للتخصيص الديناميكي. **d. جميع الإجابات السابقة صحيحة.** e. a. و c.	

54	اختر الإجابة الصحيحة: a. البرمجة التقليدية تعتمد بشكل أساسي على بنى المعطيات و الخوارزميات في كتابة برامجها. b. البرمجة التقليدية أكثر قوة ومتانة من البرمجة بالكائنات. c. البرمجة بالكائنات تعتمد بشكل أساسي على الكيانات في كتابة برامجها. d. جميع الإجابات السابقة صحيحة. **e. a و c.**
55	اختر العبارة الصحيحة: a. تخزن محتويات ملف ما في مواقع فيزيائية متتالية على القرص. b. تمكننا التعليمة التالية dir name.txt عند كتابتها في محث نظام dos من عرض محتوياته على الشاشة. c. جهاز الدخل يمكن أن يكون فقط من خلال لوحة المفاتيح keyboard. **d. جهاز الخرج القياسي يتمثل بشاشة الحاسب screen.** e. جميع الإجابات السابقة خاطئة.

2011/10/29 المهندس خالد ياسين الشيخ

معلومة خفيفة :

للتنقل بين نوافذ word المفتوحة انقر معاً Ctrl +F6 في لوحة المفاتيح.